Updatable Anonymous Credentials and Applications to Incentive Systems

Johannes Blömer Jan Bobolz Denis Diemert Fabian Eidens

February 18, 2019
Department of Computer Science
Paderborn University, Germany
{jbobolz,feidens}@mail.uni-paderborn.de

Abstract

In this paper, we introduce updatable anonymous credential systems (UACS) and use them to construct a new privacy-preserving incentive system. In a UACS, a user holding a credential certifying some attributes can interact with the corresponding issuer to update his attributes. During this, the issuer knows which update function is run, but does not learn the user's previous attributes. Hence the update process preserves anonymity of the user. One example for a class of update functions are additive updates of integer attributes, where the issuer increments an unknown integer attribute value v by some known value k. This kind of update is motivated by an application of UACS to incentive systems. Users in an incentive system can anonymously accumulate points, e.g. in a shop at checkout, and spend them later, e.g. for a discount.

In this paper, we (1) formally define UACS and their security, (2) give a generic construction for UACS supporting arbitrary update functions, and (3) construct a practically efficient incentive system using UACS.

Keywords: Anonymous Credentials, Updatable Anonymous Credentials, Privacy, Incentive System, Incentive Collection, Customer Loyalty Program

1 Introduction

In this paper we introduce updatable anonymous credential systems and use them to construct a privacy-preserving incentive system.

Updatable anonymous credential systems. Anonymous credential systems provide a privacy-preserving way of authentication in contrast to the standard authentication through identification via username and password. Authentication with identifying information allows service providers to collect and exchange user-specific data to build a comprehensive user profile without the user's consent. Anonymous credentials mitigate such problems, provide anonymity and support authentication policies [BCKL08, BBB+18, CL01, CL04, DMM+18]. A credential is parameterized with a vector of attributes (e.g., `birth date`, `affiliation`, `subscription_end` etc.) and when authenticating, users can prove possession of a credential that fulfills a certain access policy (e.g., "`affiliation = university` *`or`* `subscription_end > [today]`") without revealing anything about the attributes except that they fulfill the access policy.

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" under the project number 160364472 – SFB 901.

In an anonymous credential system, the issuer of a credential always learns the plaintext attributes of the credentials. For example, suppose that a user wants to extend a subscription for which she has a credential as described above. To extend the subscription in a traditional anonymous credential system, she would reveal all her attribute values to the issuer, who would then issue a new credential containing her old attributes and the newly updated `subscription_end` value. This means that there is no protection of privacy when updating attributes.

To solve this problem, we introduce *updatable* anonymous credential systems (UACS). An UACS has, in addition to the usual protocols of anonymous credential systems, an *update* protocol. This allows a user to interact with a credential issuer in order to update attributes in a privacy-preserving manner. More specifically, the update protocol takes as input an update function. The user contributes a hidden parameter α and her old credential with attributes $\vec{A}$. By running the protocol with the issuer, the user obtains a new credential on attributes $\vec{A}^* = \psi(\vec{A}, \alpha)$. The issuer only learns what update function ψ is applied, but does not learn $\vec{A}$ or α. In the subscription update scenario, to add 30 days to the current `subscription_end`, the update function would be $\psi((A, \mathit{subscription\_end}), \alpha) = (A, \mathit{subscription\_end} + 30)$ (in this particular case, the hidden parameter α is ignored by ψ, but we will later see update functions that depend on α).

The update functionality can be realized using only building blocks already used by most anonymous credential constructions: Zero-knowledge proofs, commitments, and blind signature schemes with efficient "signing a committed value" protocols. The idea to implement the UACS update protocol is as follows: A credential on attributes $\vec{A}$ is a digital signature on $\vec{A}$. The user prepares the update by computing $\vec{A}^* = \psi(\vec{A}, \alpha)$ and committing to $\vec{A}^*$. The user then proves that she possesses a signature on her old attributes $\vec{A}$ and that she knows α such that the commitment can be opened to $\psi(\vec{A}, \alpha)$. Afterwards, issuer and user run the blind signature protocol to jointly compute a signature on the committed $\vec{A}^*$ (i.e. the updated attributes) without revealing $\vec{A}^*$ to the issuer.

Incentive systems. A concrete application for UACS are *incentive systems*. An incentive system allows users to collect points (e.g., for every purchase they make), which they can redeem for bonus items or discounts. Such systems aim at reinforcing customer loyalty and incentivize certain behavior through points. In practice, such systems are centralized services, e.g. German Payback [PAY19] and American Express Membership Rewards program [Ame19]. In order to earn points for a purchase, the user reveals her customer ID (e.g., by showing a card). This means that the user's privacy is not protected as every purchase made can be linked to the user's identity by the incentive system provider.

To remedy this, cryptographic incentive systems [JR16, HHNR17] aim at allowing users to earn and spend points anonymously. The general idea is that users store their own points in authenticated form. We present a new construction of an incentive system based on UACS, which improves upon prior work with respect to efficiency and features (as discussed later). As a first sketch, let us assume that the user stores her point count v as an attribute in her credential. When the user earns k additional points, the incentive system provider runs an update on the user's credential, adding k points to her current point count attribute v, i.e. $\psi((v), \alpha) = (v + k)$. When the user wants to spend k points, they run an update ψ such that $\psi((v), \alpha) = v - k$ if $v \geq k$ and $\psi((v), \alpha) = \perp$ otherwise.

Of course, this first sketch does not prevent users from double-spending their points: the spend update operation creates a new credential with lowered point count, but there is no mechanism that forces the user to use the new credential. She can instead keep using the old one, which certifies a higher point count. Hence we modify the first sketch with basic double-spending protection: The attributes now include a random double-spend identifier *dsid*, i.e. attribute vectors are of the form $\vec{A} = (\mathit{dsid}, v)$. To earn points, the update function still just increases the point count ($\psi((\mathit{dsid}, v), \alpha) = (\mathit{dsid}, v + k)$). When the user wants to spend points, she reveals

her *dsid* to the provider and the provider checks that her specific *dsid* has never been revealed to it (spent) before. If that check succeeds, the user chooses a random successor double-spend identifier $dsid^*$ and sets her hidden update parameter α to $dsid^*$. User and provider then run the update $\psi((dsid, v), \alpha = dsid^*) = (dsid^*, v - k)$, embedding a new $dsid^*$ into the successor credential. If the user tries to spend her old credential (with the old *dsid*) again, the provider will detect the duplicate *dsid*. Anonymity is still preserved because $dsid^*$ is hidden from the provider until the credential is spent.

However, this approach requires all stores where points can be spent to be permanently online in order to check whether a given *dsid* has already been spent. As this is a problem in practice, *offline* double-spending protection is desirable. The idea is that stores that are offline and have an incomplete list of spent *dsid*s may incorrectly accept a spend transaction, but they can later (when they are online again) uncover the identities of double spenders. This allows the provider to recoup any losses due to offline double-spending by pursuing a legal solution to roll back illegal transactions. To incorporate offline double-spending protection, we now embed a user's secret key *usk* and a random value *dsrnd* into credentials, i.e. attributes are now $\vec{A} = (usk, dsid, dsrnd, v)$. The update function to earn points is unchanged. To spend points, the provider now sends a random challenge γ to the user and the user reveals $c = usk \cdot \gamma + dsrnd \mod p$ (where *usk*, *dsrnd* are values from her credential attributes). The user chooses new hidden random $dsid^*, dsrnd^*$ for its successor credential and then runs the update for $\psi((usk, dsid, dsrnd, v), \alpha = (dsid^*, dsrnd^*)) = (usk, dsid^*, dsrnd^*, v - k)$. As long as a credential is only spent once, *usk* is perfectly hidden in c. If the user tries to spend the same credential a second time, revealing $c' = usk \cdot \gamma' + dsrnd$ for some different challenge γ', the provider can compute *usk* from c, c', γ, γ', identifying the double-spender.

This last description comes close to the scheme we present in this paper. However, one problem remains: assume some user doubles-spends a credential $(usk, dsid, dsrnd, v)$. For both spend transaction, she receives a remainder amount credential with attributes $\vec{A}^* = (usk, dsid^*, dsrnd^*, v-k)$. While both transactions will be detected as double spending and the user's key is revealed, the user can keep using both remainder amount credentials anonymously, allowing her to spend $2 \cdot (v-k) > v$ points. To prevent this, we need a mechanism that allows us to recognize remainder amount credentials that were derived from invalid (double-spending) transactions. This can be achieved by forcing the user to reveal an encryption *ctrace* of $dsid^*$ under *usk* when spending points. As soon as a user double-spends, the issuer can compute *usk* as above. With it, he can decrypt all *ctrace* for that user, allowing him to find out what *dsid*s have been derived from invalid transactions of the double-spending user.

Related work on anonymous credential systems. There is a large body of work on anonymous credential systems, extending the basic constructions [BCKL08, CL01, CL04, PS16] with additional features such as revocation [CKS10, CL01], controlled linkability and advanced policy classes [BBB$^+$18], hidden policies [DMM$^+$18], delegation [BB18, CDD17], and many others. Our notion of privacy-preserving updates on credentials is a new feature. We show how to efficiently extend the standard blind signature based construction with updates, which makes our updates compatible with a large part of features presented in existing work (with the exception of [DMM$^+$18], which does not rely on blind signatures).

The scheme in [CKS10] allows issuers to non-interactively update credentials they have issued. In contrast to our updatable credentials, their update cannot depend on hidden attributes and the issuer learns all attributes issued or updated. Their update mechanism is mostly aimed at providing an efficient means to update revocation information, which is controlled by the issuer. Updatable credentials in the sense of our paper allow for the functionality in [CKS10] as well (although in our system, updates are done *interactively* between user and issuer). However, beyond that, our updates can depend on hidden attributes of the user and the issuer does not learn the attributes resulting from the update.

More technically similar to our updatable credential mechanism are *stateful* anonymous credentials [CGH11]. A stateful credential contains a state. The user can have his credential state updated to some successor state as prescribed by a public state machine model. For this, the user does not have to disclose his current credential state. Such a state transition is a special case of an update to a state attribute in an updatable credential. In this sense, our construction of updatable credentials generalizes the work of [CGH11].

Related work on incentive systems. Existing e-cash systems are related to incentive systems focused on point collection, but pursue different security goals [CHL05]. E-cash does not support the accumulation of points within a single token. Instead, each token corresponds to a coin and can be identified. To spend a coin, a user transfers it to another owner. In incentive systems, a token accumulates a number of points into a single token (i.e. the token is like a bank account rather than a coin).

A cryptographic scheme that considers the collection of points in a practical scenario is described by Milutinovic et al. in [MDPD15]. Their scheme uCentive can be seen as a special e-cash system where a so called uCent corresponds to a point. The user stores and spends all uCents individually, which induces storage and communication cost linear in the number of uCents. Similar to our system, uCentive builds upon anonymous credentials (but without updates) and commitments, but to detect double-spending the issuer has to be online.

Jager and Rupp [JR16] introduce black-box accumulation (BBA) as a building-block for incentive systems. They formalize the core functionality and security of such systems based on the natural requirement that users collect and sum up values in a privacy-preserving way. In detail, they present an generic construction of BBA from homomorphic commitments, digital signatures, and non-interactive zero-knowledge proofs of knowledge (Goth-Sahai proofs [GS08]). The BBA solution has three major shortcomings: the token creation and redemption processes are linkable, users have to redeem all of their points at once, and stores must be permanently online to detect double-spending.

Hartung et al. [HHNR17] present an improved framework of black-box accumulation (BBA+) based on the framework introduced in [JR16]. In [HHNR17], BBA is extended with offline double-spending prevention (on which we base our offline double-spending mechanism) and other desirable features. Because of efficiency reasons, when spending points, the user needs to reveal her point count. This is due to the use of Groth-Sahai proofs, which makes range proofs of the form $v \geq k$ while hiding v very costly. In contrast, our incentive system can be instantiated in a Schnorr proof setting, making range proofs much more viable [BBB$^+$18, CCs08]. For this reason, our incentive system can hide the user's point count without taking too much of a performance hit.

What prior work does not handle is the scenario in which the user spends *some* of her points at an offline store and is issued a new token for the remainder point value. If the spend operation is later detected double-spending, the remainder point token still remains valid in prior constructions. Our construction solves this, allowing the issuer to trace all tokens derived from illegal (double-spent) transactions. The price of this solution is forward and backward privacy as defined in [HHNR17], which our scheme does not offer.

Our contribution and structure of this paper. We introduce UACS formally in Section 3 and define its security properties. In Section 4, we construct UACS generically from blind signature schemes. We define formal requirements for incentive systems in Section 5, modeling our double-spend prevention mechanism and defining security. In Section 6, we construct an incentive system from a UACS. Finally, we practically evaluate our incentive system in Section 7.

2 Preliminaries

Throughout the paper, we refer to a public-parameter generation ppt $\mathcal{G}$ that outputs public parameters pp given unary security parameter 1^λ. A function $f : \mathbb{N} \to \mathbb{R}$ is negligible if for all $c > 0$ there is a x_0 such that $f(x) < 1/x^c$ for all $x > x_0$. We refer to a negligible function as *negl*. We write $\text{output}_A[A \leftrightarrow B]$ for interactive algorithms A, B to denote the output of A after interacting with B. The support of a probabilistic algorithm A on input x is denoted by $[A(x)] = \{y \mid \Pr[A(x) = y] > 0\}$. The expression $\mathsf{ZKAK}[(w); (x, w) \in R]$ denotes a zero-knowledge argument of knowledge protocol where the prover proves knowledge of w such that (x, w) is in some NP relation R. The zero-knowledge argument of knowledge can be simulated perfectly given a trapdoor [Dam00] and there exists an expected polynomial-time extractor that, given black-box access to a successful prover, computes a witness w with probability 1 [Dam00].

For blind signatures, we require that the blind signing protocol is of the form "commit to the message(s) to sign, then jointly compute the signature". In our formulation, we externalize the ZK proof about the commitment (cf. Definition 18).

Definition 1. A blind signature scheme for signing committed values $\Pi_\mathcal{S}$ consists of the following (ppt) algorithms:

$\mathsf{KeyGen}_\mathcal{S}(pp, 1^n) \to (pk, sk)$ generates a key pair (pk, sk) for signatures on vectors of n messages. We assume n can be efficiently derived from pk.

$\mathsf{Commit}_\mathcal{S}(pp, pk, \vec{m}, r) \to c$ given messages $\vec{m} \in \mathcal{M}^n$ and randomness r, deterministically computes a commitment c.

$\mathsf{BlindSign}_\mathcal{S}(pp, pk, sk, c) \leftrightarrow \mathsf{BlindRcv}_\mathcal{S}(pp, pk, \vec{m}, r) \to \sigma$ is an interactive protocol with common input pp, pk. The signer's input is sk, c. The receiver's input consists of the messages $\vec{m}$ and commitment randomness r. The receiver outputs a signature σ or the error symbol $\perp$.

$\mathsf{Vrfy}_\mathcal{S}(pp, pk, \vec{m}, \sigma) \to b$ deterministically checks signature σ and outputs 0 or 1.

A blind signature scheme is *correct* if for all $\lambda, n \in \mathbb{N}$ and all $pp \in [\mathcal{G}(1^\lambda)]$, $(pk, sk) \in [\mathsf{KeyGen}(pp, 1^n)]$, all $\vec{m} \in \mathcal{M}^n$, and for every commitment randomness r,

$$\Pr[\mathsf{BlindSign}(pp, pk, sk, \mathsf{Commit}(pp, pk, \vec{m}, r)) \leftrightarrow \mathsf{BlindRcv}(pp, pk, \vec{m}, r) \to \sigma : \mathsf{Vrfy}(pp, pk, \vec{m}, \sigma) = 1] = 1$$

$\diamond$

We require unforgeability and perfect message privacy, cf. Appendix A, Definition 18 and 19. Definition 1 can be instantiated by Pointcheval Sanders signatures [PS16].

Definition 2. A public-key enc scheme $\Pi_\mathcal{E}$ consists of ppt algorithms $(\mathsf{KeyGen}_\mathcal{E}, \mathsf{ComputePK}_\mathcal{E}, \mathsf{Encrypt}_\mathcal{E}, \mathsf{Decrypt}_\mathcal{E})$ such that $\mathsf{Decrypt}_\mathcal{E}$ and $\mathsf{ComputePK}_\mathcal{E}$ are deterministic and for all $pp \in [\mathcal{G}(1^\lambda)]$, $sk \in [\mathsf{KeyGen}_\mathcal{E}(pp)]$ and all messages m, it holds that $\Pr[\mathsf{Decrypt}_\mathcal{E}(pp, sk, \mathsf{Encrypt}(pp, \mathsf{ComputePK}_\mathcal{E}(pp, sk), m)) = m] = 1$.

We require key-indistinguishable CPA security, cf. Appendix A, Definition 20. Definition 2 can be instantiated by ElGamal encryption.

Definition 3. A malleable commitment scheme $\Pi_{\mathcal{C}^+}$ consists of ppt algorithms $(\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Vrfy}, \mathsf{Add})$ s.t. $\forall pp \in [\mathcal{G}(1^\lambda)]$, $pk \in [\mathsf{KeyGen}(pp)]$, all $m \in M_{pp}$, and all $(c, o) \in [\mathsf{Commit}(pp, pk, m)]$

- M_{pp} is an (additive) group.

- Vrfy and Add are deterministic.
- $\mathsf{Vrfy}(pp, pk, c, o, m) = 1$.
- For $c' = \mathsf{Add}(pp, pk, c, k)$, it holds that $(c', o) \in [\mathsf{Commit}(pp, pk, m + k)]$.

We require perfect binding and comp. hiding, cf. Appendix A, Definition 21 and 22. Definition 3 can be instantiated by ElGamal encryption.

3 Updatable Anonymous Credentials

In this section, we introduce updatable anonymous credentials. In UACS, there are three roles: issuers, users, and verifiers. Issuers hold keys to issue and update credentials for users. Users can prove possession of a credential to verifiers.

We generalize the credential issuing protocol: usually, the issue operation takes attributes $\vec{A}$ as input, meaning that the issuer knows exactly what attributes he is issuing. In our generalization, the issue operation takes an update function ψ as input. The user may choose private input α. The credential is obtained on $\vec{A} = \psi(\perp, \alpha)$ (one can think of issuing a new credential as an "update" of an empty credential). We stress that this is a generalization and that issuers can choose ψ to output a constant.

Usually a user's secret key *usk* is embedded into the credential to bind the credential to the user [CL04]. This is a possibility covered by our generalization (treat *usk* like any other hidden attribute). For this reason, we omit user keys (and pseudonyms) from our definitions and allow the application layer to implement them in any desired way (like our incentive system (Construction 14) does).

Definition 4. An updatable anonymous credential system (UACS) $\Pi_{\mathcal{C}}$ consists of the following ppt algorithms:

$\mathsf{Setup}(pp) \to cpp$ generates credential public parameters *cpp*. We assume an attribute universe $\mathbb{A}$ to be encoded in *cpp*.

$\mathsf{IssuerKeyGen}(cpp, 1^n) \to (pk, sk)$ generates a credential issuer key pair (pk, sk) for credentials with n attributes. We assume that n can be derived from pk and that cpp, pk fix an update function universe Ψ and a show predicate universe Φ.

$\mathsf{Issue}(cpp, pk, \psi, sk) \leftrightarrow \mathsf{Receive}(cpp, pk, \psi, \alpha) \to cred$ is an interactive protocol with common input cpp, pk and update function $\psi \in \Psi, \psi : \{\perp\} \times \{0,1\}^* \to \mathbb{A}^n \cup \{\perp\}$. The issuer gets its secret key sk as private input and the receiver gets its secret update function input $\alpha \in \{0,1\}^*$ as private input. After the protocol, the receiver outputs a credential *cred* (on $\psi(\perp, \alpha)$) or the failure symbol $\perp$.

$b \leftarrow \mathsf{Update}(cpp, pk, \psi, sk) \leftrightarrow \mathsf{UpdRcv}(cpp, pk, \psi, \alpha, cred) \to cred^*$ is an inter. protocol with common input *cpp*, the credential issuer's pk, and update function $\psi \in \Psi, \psi : \mathbb{A}^n \times \{0,1\}^* \to \mathbb{A}^n \cup \{\perp\}$. As private input, the issuer gets sk and the receiver gets $\alpha \in \{0,1\}^*$ and credential-to-be-updated *cred*. The receiver outputs a new credential $cred^*$ or $\perp$. The issuer outputs a bit b.

$\mathsf{ShowPrv}(cpp, pk, \phi, cred) \leftrightarrow \mathsf{ShowVrfy}(cpp, pk, \phi) \to b$ is an inter. protocol with common input *cpp*, the issuer's pk, and a statement over attributes $\phi \in \Phi, \phi : \mathbb{A}^n \to \{0,1\}$. The prover gets *cred* as input. The verifier outputs a bit b.

We define correctness by defining a correctness set S recursively as follows: Let $\lambda, n \in \mathbb{N}$, $cpp \in [\mathsf{Setup}(\mathcal{G}(1^\lambda))]$, $(pk, sk) \in [\mathsf{IssuerKeyGen}(cpp, 1^n)]$,

- Let $\psi \in \Psi, \alpha \in \{0,1\}^*$. If *cred* is possible output of $\mathsf{Issue}(cpp, pk, \psi, sk) \leftrightarrow \mathsf{Receive}(cpp, pk, \psi, \alpha)$, then $(cpp, pk, \psi(\bot, \alpha), cred) \in S$.
- If $(cpp, pk, \vec{A}, cred) \in S$ and $\psi \in \Psi$, $\alpha \in \{0,1\}^*$ with $\psi(\vec{A}, \alpha) \neq \bot$, and $cred^*$ is possible output of $\mathsf{Update}(cpp, pk, \psi, sk) \leftrightarrow \mathsf{UpdRcv}(cpp, pk, \psi, \alpha, cred) \rightarrow cred^*$, then $(cpp, pk, \psi(\vec{A}, \alpha), cred^*) \in S$.

For all $(cpp, pk, \vec{A}, cred) \in S$ and $\phi \in \Phi$ with $\phi(\vec{A}) = 1$, it holds that $\Pr[\mathsf{ShowPrv}(cpp, pk, \phi, cred) \leftrightarrow \mathsf{ShowVrfy}(cpp, pk, \phi) \rightarrow b : \ b = 1] = 1$, and for all $\psi \in \Psi, \alpha \in \{0,1\}^*$ with $\psi(\vec{A}, \alpha) \neq \bot$, $\Pr[b \leftarrow \mathsf{Update}(cpp, pk, \psi, sk) \leftrightarrow \mathsf{UpdRcv}(cpp, pk, \psi, \alpha, cred) : \ b = 1] = 1$. ◇

The system is set up using $cpp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda))$, run by a trusted party. The credential public parameters *cpp* are then published. An issuer is set up by running $\mathsf{IssuerKeyGen}()$. Issuers can issue credentials by running Issue interacting with a user running Receive. To update a credential, its issuer has Update interact with a user's UpdRcv. To prove possession of a credential, a user runs ShowPrv and any verifier runs ShowVrfy (no special key needed for the verifier).

Security consists of two properties: *anonymity*, which protects users' privacy, and *soundness*, which protects issuers and verifiers. For anonymity, we require that the user's protocols can be simulated without private inputs.

Definition 5. Let $\Pi_{\mathcal{C}}$ be an UACS and let $\mathsf{CheckCred}(cpp, pk, cred, \vec{A}) = 1$ iff $cred \neq \bot$ is possible output of Receive or UpdRcv with ψ, α that results in $\vec{A}$. Let $\mathsf{Receive}'$ run $cred \leftarrow \mathsf{Receive}$ and then send a bit indicating whether or not $cred = \bot$. Let UpdRcv' run $cred^* \leftarrow \mathsf{UpdRcv}$ and then send a bit indicating whether or not $cred^* = \bot$. Π has *simulation anonymity* if there exist ppt simulators $\mathfrak{S}_{\mathsf{Setup}}, \mathfrak{S}_{\mathsf{Receive}}, \mathfrak{S}_{\mathsf{ShowPrv}}, \mathfrak{S}_{\mathsf{UpdRcv}}$ such that for all (unrestricted) adversaries $\mathcal{A}$, all $\lambda \in \mathbb{N}$, $pp \in [\mathcal{G}(1^\lambda)]$, and all $(cpp, td) \in [\mathfrak{S}_{\mathsf{Setup}}(pp)]$,

- $\Pr[\mathsf{Setup}(pp) = cpp] = \Pr[\mathfrak{S}_{\mathsf{Setup}}(pp) = (cpp, \cdot)]$
- $\mathrm{output}_{\mathcal{A}}[\mathfrak{S}_{\mathsf{Receive}}(td, pk, \psi) \leftrightarrow \mathcal{A}] \approx \mathrm{output}_{\mathcal{A}}[\mathsf{Receive}'(cpp, pk, \psi, \alpha) \leftrightarrow \mathcal{A}]$ for all $pk, \alpha \in \{0,1\}^*, \psi \in \Psi$.
- $\mathrm{output}_{\mathcal{A}}[\mathfrak{S}_{\mathsf{UpdRcv}}(td, pk, \psi) \leftrightarrow \mathcal{A}] \approx \mathrm{output}_{\mathcal{A}}[\mathsf{UpdRcv}'(cpp, pk, \psi, \alpha, cred) \leftrightarrow \mathcal{A}]$ for all pk, $\psi \in \Psi$ and $cred, \vec{A}$ s.t. $\mathsf{CheckCred}(cpp, pk, cred, \vec{A}) = 1$ and $\psi(\vec{A}, \alpha) \neq \bot$.
- $\mathrm{output}_{\mathcal{A}}[\mathfrak{S}_{\mathsf{ShowPrv}}(td, pk, \phi) \leftrightarrow \mathcal{A}] \approx \mathrm{output}_{\mathcal{A}}[\mathsf{ShowPrv}(cpp, pk, \phi, cred) \leftrightarrow \mathcal{A}]$ for all pk, $\phi \in \Phi$ and $cred, \vec{A}$ s.t. $\mathsf{CheckCred}(cpp, pk, cred, \vec{A}) = 1$ and $\phi(\vec{A}) = 1$ ◇

Informally, *soundness* should enforce that users cannot show credentials they have not been issued. In UACS, the issuer does not know what attributes result from Issue and Update operations, so the issuer/verifier cannot easily check whether or not security was broken. For this reason, our soundness definition requires existence of an extractor $\mathcal{E}$ that outputs an *explanation list* $\mathcal{L}$. $\mathcal{L}$ lists hidden parameters describing, for example, what $(\vec{A}, \alpha)$ went into an update operation. The adversary wins if $\mathcal{L}$ is not consistent, e.g., if the claimed $\vec{A}$ of an update operation has not been the result of some earlier operation.

Definition 6. Let $\mathcal{E}, \mathcal{A}$ be algorithms. We define the experiment $\mathrm{Exp}^{\mathrm{sound}}$ in Fig. 1. We say that Π is *sound* if there exists an algorithm $\mathcal{E}$, running in expected polynomial time, such that for all ppt adversaries $\mathcal{A}$, there exists a negligible function *negl* with $\Pr[\mathrm{Exp}^{\mathrm{sound}}(\Pi, \mathcal{A}, \mathcal{E}, \lambda) = 1] \leq negl(\lambda)$ for all λ. ◇

$\mathrm{Exp}^{\mathrm{sound}}(\Pi, \mathcal{A}, \mathcal{E}, \lambda)$:
$cpp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda))$, $(1^n, st) \leftarrow \mathcal{A}(cpp)$, for some $n \in \mathbb{N}$
Setup up issuer: $(pk, sk) \leftarrow \mathsf{IssuerKeyGen}(cpp, 1^n)$
$\mathsf{halt} \leftarrow \mathcal{A}(pk, st)^{\mathsf{oracles}}$
where oracles gives access:
- To receive credentials, $\mathcal{A}$ specifies $\psi \in \Psi$. Run $\mathsf{Issue}(cpp, pk, \psi, sk)$ interacting with $\mathcal{A}$. Append the randomness used for Issue to r_{Issue}.
- To update credentials, $\mathcal{A}$ specifies $\psi \in \Psi$. Run $\mathsf{Update}(cpp, pk, \psi, sk) \to b$ interacting with $\mathcal{A}$. Append the randomness used for Update to r_{Update}.
- To show a credential, $\mathcal{A}$ specifies $\phi \in \Phi$. Run $\mathsf{ShowVrfy}(cpp, pk, \phi) \to b$ interacting with $\mathcal{A}$. Append the randomness used for ShowVrfy to r_{ShowVrfy}.

Let $r_{\mathcal{A}}$ be the randomness used by $\mathcal{A}$.
Run $\mathcal{E}^{\mathcal{A}}(cpp, r_{\mathcal{A}}, r_{\mathsf{Issue}}, r_{\mathsf{Update}}, r_{\mathsf{ShowVrfy}})$ to receive an *explanation list* $\mathcal{L}$:
Each entry on $\mathcal{L}$ corresponds to an issue, update, or show query made by $\mathcal{A}$.
We set $E_0 = \emptyset$. E_i represents the set of explained attribute vectors after the ith query.
We say that $\mathcal{L}$ is *consistent* if there are sets $E_1, E_2, \ldots$ such that for all $i > 0$:
- If the ith operation was an Issue operation with update function ψ_i, then the ith entry on the list is some $\alpha_i \in \{0,1\}^*$. $E_i = E_{i-1} \cup \{\psi_i(\perp, \alpha_i)\} \setminus \{\perp\}$.
- If the ith operation was an Update operation with update function ψ_i, then the ith entry on the list is a tuple $(\vec{A}_i, \alpha_i)$. If Update output 1, then $\psi_i(\vec{A}_i, \alpha_i) \neq \perp$ and $\vec{A}_i \in E_{i-1}$ and $E_i = E_{i-1} \cup \{\psi_i(\vec{A}_i, \alpha_i)\}$. Otherwise, $E_i = E_{i-1}$.
- If the ith operation was a ShowVrfy operation with predicate ϕ_i, then the ith entry on the list is an attribute vector $\vec{A}_i$. If ShowVrfy output 1, then $\phi(\vec{A}_i) = 1$ and $\vec{A}_i \in E_{i-1}$. $E_i = E_{i-1}$.

Output 0 if $\mathcal{E}$'s list is consistent
return 1

Figure 1: Soundness experiment for updatable anonymous credential systems

4 Generic Construction of UACS

We construct a UACS generically from blind signatures. The construction is simple: As usual in credential systems, a credential is a signature on the credential's attributes. To update a credential with update function ψ, the user commits to the attributes $\vec{A}^* = \psi(\vec{A}, \alpha)$ that it wants to receive. He then proves within a zero-knowledge argument of knowledge that this commitment is correctly formed and that he already possesses a credential (signature) on $\vec{A}$. Then, the issuer blindly signs the committed value $\vec{A}^*$, resulting in a new credential.

Construction 7. Let $\Pi_{\mathcal{S}}$ be a blind signature (Definition 1). We construct UACS:

$\mathsf{Setup}(pp) \to cpp$ generates public parameters cpp consisting of pp and a zero-knowledge argument common reference string. The attribute space $\mathbb{A}$ is the signature scheme's message space $\mathcal{M}_{\mathcal{S}}$.

$\mathsf{IssuerKeyGen}(cpp, 1^n) \to (pk, sk)$ runs $\mathsf{KeyGen}_{\mathcal{S}}(pp, 1^{n+1}) \to (pk, sk)$. The update function universe Ψ consists of all $\psi : (\mathcal{M}_{\mathcal{S}}^n \cup \{\perp\}) \times \{0,1\}^* \to \mathcal{M}_{\mathcal{S}}^n \cup \{\perp\}$ that are supported by the zero-knowledge arguments below.

$\mathsf{Issue}(cpp, pk, \psi, sk) \leftrightarrow \mathsf{Receive}(cpp, pk, \psi, \alpha) \to cred$ for $\psi \in \Psi$ works as follows:

- The receiver computes $\vec{A} = \psi(\perp, \alpha)$ and commits to $\vec{A}$ by computing $c = \mathsf{Commit}_{\mathcal{S}}(pp, pk, \vec{A}, r)$ for random r and sends c to the issuer.
- The receiver proves $\mathsf{ZKAK}[(\alpha, r); c = \mathsf{Commit}_{\mathcal{S}}(pp, pk, \psi(\perp, \alpha), r)]$
- If the proof accepts, issuer runs $\mathsf{BlindSign}_{\mathcal{S}}(pp, pk, sk, c)$ and receiver runs $\mathsf{BlindRcv}_{\mathcal{S}}(pp, pk, \vec{A}, r) \to \sigma$.
- The receiver checks if $\mathsf{Vrfy}_{\mathcal{S}}(pp, pk, \vec{A}, \sigma) = 1$. If so, it outputs $cred = (\vec{A}, \sigma)$, otherwise it outputs $\perp$.

$b \leftarrow \mathsf{Update}(cpp, pk, \psi, sk) \leftrightarrow \mathsf{UpdRcv}(cpp, pk, \psi, \alpha, cred) \to cred^*$:

- The receiver parses $cred = (\vec{A}, \sigma)$ and computes $\vec{A}^* = \psi(\vec{A}, \alpha)$.

- The receiver commits to $\vec{A}^*$ by computing $c = \mathsf{Commit}_{\mathcal{S}}(pp, pk, \vec{A}^*, r)$ for random r and sends c to the issuer.
- The receiver proves $\mathsf{ZKAK}[(\vec{A}, \sigma, \alpha, r); c = \mathsf{Commit}_{\mathcal{S}}(pp, pk, \psi(\vec{A}, \alpha), r) \wedge \mathsf{Vrfy}_{\mathcal{S}}(pp, pk, \vec{A}, \sigma) = 1]$.
- If the proof rejects, the issuer outputs 0 and aborts.
- Otherwise, issuer runs $\mathsf{BlindSign}_{\mathcal{S}}(pp, pk, sk, c)$ while receiver runs $\mathsf{BlindRcv}_{\mathcal{S}}(pp, pk, \vec{A}^*, r) \to \sigma^*$.
- The receiver checks if $\mathsf{Vrfy}_{\mathcal{S}}(pp, pk, \vec{A}^*, \sigma^*) = 1$. If so, it outputs $cred^* = (\vec{A}^*, \sigma^*)$, otherwise it outputs $\perp$. The issuer outputs 1.

$\mathsf{ShowPrv}(cpp, pk, \phi, cred) \leftrightarrow \mathsf{ShowVrfy}(cpp, pk, \phi) \to b$ works as follows: the prover parses $cred = (\vec{A}, \sigma)$. If $\phi(\vec{A}) = 0$, the prover aborts and the verifier outputs 0. Otherwise, the prover runs the proof $\mathsf{ZKAK}[(\vec{A}, \sigma); \mathsf{Vrfy}_{\mathcal{S}}(pp, pk, \vec{A}, \sigma) = 1 \wedge \phi(\vec{A}) = 1]$. If the proof succeeds, the verifier outputs 1, otherwise 0.

Correctness of the construction follows directly from the correctness of the underlying blind signature scheme. Since there exist zero-knowledge arguments of knowledge for all of NP, almost arbitrary update functions are supported by this construction. Because those generic zero-knowledge arguments are not necessarily considered practically efficient, in practice one would usually restrict the class of update functions. For example, a large class of statements is supported by Sigma protocols (such as generalizations of Schnorr's protocol), which are very efficient (see, for example, [BBB+18]). The blind signature scheme by Pointcheval and Sanders [PS16] is a good candidate to use in conjunction with Sigma protocols. If the update function is sufficiently "simple" (i.e. the check $\psi(\vec{A}, \alpha) \stackrel{!}{=} \vec{A}^*$ can be efficiently implemented as a Sigma protocol), our construction is efficient.

Theorem 8. If the underlying blind signature scheme has perfect message privacy (Definition 19), then Const. 7 has simulation anonymity (Definition 5).

Theorem 9. If the underlying blind signature scheme is unforgeable (Definition 18), then Const. 7 is sound (Definition 6).

The proofs of the above theorems are straight-forward reductions to the corresponding blind signature properties. They are presented in Appendix B.

5 Incentive Systems

In this section, we formally define incentive systems, their syntax, and their security. In an incentive system, there are two roles: users and issuers. Users accumulate points issued by an issuer in a privacy-preserving way. In the following we formalize our notion of incentive systems for earning and spending points. Additionally, we define linking and tracing for double-spending prevention.

Definition 10. An inc. system Π_{InSy} consists of the following ppt algorithms:

$\mathsf{Setup}(pp) \to ispp$ generates $ispp$. We assume a maximum point score v_{max} is encoded in $ispp$ (we assume this limit to be large enough never to be hit in practice).

$\mathsf{KeyGen}(ispp) \to (upk, usk)$ generates a user's public key upk, and secret key usk.

$\mathsf{IssuerKeyGen}(ispp) \to (pk, sk)$ generates an issuer key pair (pk, sk).

$\mathsf{Issue}(ispp, pk, upk, sk) \leftrightarrow \mathsf{Join}(ispp, pk, upk, usk) \to (token, dsid)$ The receiver outputs a token $token$ with 0 points and a double-spend id $dsid$, or $\perp$.

$\mathsf{Credit}(ispp, pk, k, sk) \leftrightarrow \mathsf{Earn}(ispp, pk, k, usk, token) \rightarrow token^*$ takes common input earn amount $k \in \mathbb{N}$. The earner outputs an updated token $token^*$ or $\perp$.

$(token^*, dsid^*) \leftarrow \mathsf{Spend}(ispp, pk, k, dsid, usk, token) \leftrightarrow \mathsf{Deduct}(ispp, pk, k, dsid, sk) \rightarrow (b, dstag)$ takes common input spend amount k, and double-spend id $dsid$. The verifier outputs a bit b and a double-spend tag $dstag$. The spender outputs an updated token $token^*$ for the remaining amount.

$\mathsf{Link}(ispp, dstag, dstag') \rightarrow (upk, dslink)$ given double-spend tags $dstag, dstag'$, deterministically outputs the spending user's upk and linking information $dslink$.

$\mathsf{VrfyDs}(ispp, dslink, upk) \rightarrow b$ given $dslink$ and upk, outputs a bit b indicating whether $dslink$ proves that upk has double-spent.

$\mathsf{Trace}(ispp, dslink, dstag) \rightarrow dsid^*$ given $dslink$ and some spend-operation's $dstag$, deterministically outputs the remainder token's $dsid^*$.

For correctness, let $\lambda \in \mathbb{N}$. Consider $\ell \leq poly(\lambda)$ users with the following keys $((upk_1, usk_1), \ldots, (upk_\ell, usk_\ell)) \leftarrow \mathsf{KeyGen}(ispp)$, where $ispp \in [\mathsf{Setup}(\mathcal{G}(1^\lambda))]$. Let $(token_i, dsid_i)$ be the receiver's output of the execution of $\mathsf{Issue}(ispp, pk, upk_i, sk) \leftrightarrow \mathsf{Join}(ispp, pk, upk_i, usk_i)$. Let $v_i = 0$ and $T = \emptyset$ initially. Consider some sequence of operations in the form

- Update $token_i$ with $token_i \leftarrow \mathsf{Earn}(ispp, pk, k, usk, token_i) \leftrightarrow \mathsf{Credit}(ispp, pk, k, sk)$ for $k \leq v_{max} - v_i$ and update $v_i \leftarrow v_i + k$.
- Run $(token_i^*, dsid_i^*) \leftarrow \mathsf{Spend}(ispp, pk, k, dsid_i, usk, token_i) \leftrightarrow \mathsf{Deduct}(ispp, pk, k, dsid_i, sk) \rightarrow (b, dstag)$ for $k \leq v_i$. We say that *the spending failed* if $b = 0$ or $dsid \in T$. Afterwards, update $v_i \leftarrow v_i - k$, $T \leftarrow T \cup \{dsid\}$, $(token_i, dsid_i) \leftarrow (token_i^*, dsid_i^*)$.

The scheme is correct if there exists a negligible function $negl$ such that after *any* such sequence of polynomial length, the probability that any spending operation within this sequence fails is negligible (where the probability is over the random bits of $\mathsf{KeyGen}, \mathsf{Earn}, \mathsf{Credit}, \mathsf{Spend}, \mathsf{Deduct}$). $\diamond$

$\mathsf{Setup}(\mathcal{G}(\cdot))$ is run by a trusted party. $\mathsf{IssuerKeyGen}$ is run by the issuer. Each user first runs KeyGen and then joins the issuer's system using Join. Users can Earn and Spend points. The issuer uses $\mathsf{Link}, \mathsf{Trace}$ to deal with double-spending (as described below). VrfyDs is used by a judge to verify that a user has double-spent.

In order to detect double-spending offline, the issuer maintains a database $\mathcal{DB}$ and (eventually, in online phases) inserts observed transactions into that database. The database is a directed bipartite graph with two types of nodes: token nodes and spend transaction nodes. Token nodes are double-spend identifiers $dsid$. Token nodes whose owner the issuer has uncovered are associated with values $(upk, dslink)$. Spend transaction nodes t_i are associated with $k, dstag$ observed during the operation by the issuer, and a validity flag. Spend operations t_i have in-degree 1 (corresponding to the $dsid$ that was input to the operation) and out-degree at most 1 (corresponding to $dsid^*$ of the remainder token issued in the spend operation, if known to the issuer). The idea is that the database keeps a record of all $dsid$ known to it and marks transactions invalid that (1) are not the first to spend a specific $dsid$, or (2) are derived from some double-spending transaction's remainder token (which should never have been issued as the double-spend transaction should never have happened). We imagine that the provider uses lawsuits or similar mechanisms to recoup any loss that may have resulted from invalid transactions.

More concretely, we formalize tracing behavior with an algorithm DBsync that is called once for each observed spend transaction (not necessarily in the order the transactions happen). DBsync takes as input spend operation parameters $k, dsid, dstag$ and operates on the graph $\mathcal{DB}$ mentioned above as follows:

DBsync$(k, dsid, dstag, \mathcal{DB})$**:**

- Add a new spend operation node t_i to $\mathcal{DB}$ and associate $k, dstag$ with it.
- If $dsid$ is not a node in $\mathcal{DB}$, add the node $dsid$ and an edge from $dsid$ to t_i.
- Otherwise, add the edge from $dsid$ to t_i, and:
  - If $dsid$ has no $(upk, dslink)$ associated with it, then there exist two outgoing edges from $dsid$ to transactions t_i, t_j. In this case, compute $(upk, dslink) = \mathsf{Link}(ispp, dstag'_i, dstag'_j)$ using the two tags $dstag'_i, dstag'_j$ associated with t_i and t_j, respectively. Associate $(upk, dslink)$ with $dsid$.
  - Mark t_i invalid (this triggers the steps below).
- Whenever some node t_i with incoming edge from some $dsid$ is marked invalid
  - Use $(upk, dslink)$ associated with $dsid$ and $dstag$ associated to t_i to compute $dsid^* = \mathsf{Trace}(ispp, dslink, dstag)$. Add $dsid^*$ to the graph (if it does not already exist), associate $(upk, dslink)$ with $dsid^*$, and add an edge from t_i to $dsid^*$. If there is an edge from $dsid^*$ to some t_j, mark t_j invalid (if it was not already marked). This triggers this routine again.

An example database with one run of DBsync is shown in Figure 2. For invalid transaction t_i, user upk (according to the predecessor $dsid$ of t_i) is blamed.

To define security for incentive systems, we first define several stateful oracles, a selection of which will be available to an adversary $\mathcal{A}$ in each of the subsequent security games. For these definitions, we assume that $ispp$ has been generated honestly. Some oracles are interactive, i.e. they may send and receive messages during a call. We distinguish between the oracle's (local) *output*, which is generally given to the adversary, and the oracle's *sent and received messages*. The notation $x \mapsto \mathbf{Oracle}(x) \leftrightarrow \mathcal{A}$ denotes that $\mathcal{A}$ chooses x, then oracle $\mathbf{Oracle}(x)$ is run interacting with $\mathcal{A}$. $\mathcal{A}$ is given the output of the oracle (if any). The notation $(x, y) \mapsto \mathbf{Oracle}_0(x) \leftrightarrow \mathbf{Oracle}_1(y)$ denotes that $\mathcal{A}$ chooses x, y, then $\mathbf{Oracle}_0(x) \leftrightarrow \mathbf{Oracle}_1(y)$ are run, interacting with one another. $\mathcal{A}$ is given the output of both oracles, but not the messages sent or received by the oracles.

Honest users. To model honest users, we define the following oracles:

- **Keygen**() chooses a fresh user handle u, runs $(upk, usk) \leftarrow \mathsf{KeyGen}(ispp)$, and stores $(upk_u, usk_u, v_u, pk_u, token_u, dsid_u) \leftarrow (upk, usk, 0, \perp, \perp, \perp)$. It outputs u, upk.
- **Join**(u, pk) given handle u runs $(token, dsid) \leftarrow \mathsf{Join}(ispp, pk, upk_u, usk_u)$. If $token = \perp$, the oracle outputs $\perp$. Otherwise, it stores $pk_u \leftarrow pk$, $token_u \leftarrow token$, and $dsid_u \leftarrow dsid$. This oracle can only be called once for each u. It must be called before any calls to **Earn**$(u, \cdot)$ and **Spend**$(u, \cdot)$.
- **Earn**(u, k) given handle u and $k \in \mathbb{N}$ with $v_u + k \leq v_{max}$, the oracle runs $token^* \leftarrow \mathsf{Earn}(ispp, pk_u, k, usk_u, token_u)$. If $token^* = \perp$, the oracle outputs $\perp$. Otherwise, it updates $token_u \leftarrow token^*$ and $v_u \leftarrow v_u + k$.
- **Spend**(u, k) given handle u and $k \in \mathbb{N}$ with $v_u \geq k$, the oracle first sends $dsid_u$ to its communication partner and then runs $(token^*, dsid^*) \leftarrow \mathsf{Spend}(ispp, pk_u, k, dsid_u, usk_u, token_u)$. It

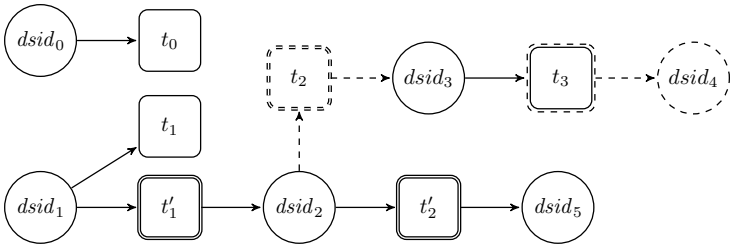

Figure 2: Example $\mathcal{DB}$. Double-struck spend operations are invalid. All dashed lines are added when t_2 is synchronized into $\mathcal{DB}$. The user has double-spent $dsid_1$ (and t'_1 is marked invalid because of this). When t_2 is synchronized into $\mathcal{DB}$, it is immediately marked invalid, $dsid_3$ is revealed to be its successor and as a consequence, t_3 is marked invalid and its successor $dsid_4$ is computed.

$\mathrm{Exp}_b^{\text{anon-X}}(\Pi_{\text{InSy}}, \mathcal{A}, \lambda)$ for
$X \in \{\mathsf{Earn}, \mathsf{Spend}\}, b \in \{0,1\}$:
$ispp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda))$
$(pk, st) \leftarrow \mathcal{A}(ispp)$
$(u_0, u_1, k, st) \leftarrow \mathcal{A}^{\mathsf{oracle}}(st)$
where $\mathsf{oracle} = (\mathbf{Keygen}(), \mathbf{Join}(\cdot, pk),$
$\mathbf{Earn}(\cdot,\cdot), \mathbf{Spend}(\cdot,\cdot))$
Output 0 if $\perp \in \{token_{u_0}, token_{u_1}\}$
Challenge Phase:
If $X = \mathsf{Earn}$ and $k \in \mathbb{N}$ with
$v_{u_0} + k \leq v_{max}$ and $v_{u_1} + k \leq v_{max}$
$\hat{b} \leftarrow \mathcal{A}^{\mathbf{Earn}(u_b,k)}(st)$
If $X = \mathsf{Spend}$ and $k \in \mathbb{N}$ with
$v_{u_0} \geq k$ and $v_{u_1} \geq k$
$\hat{b} \leftarrow \mathcal{A}^{\mathbf{Spend}(u_b,k)}(st)$
Output 1 if $b = \hat{b}$, otherwise output 0

$\mathrm{Exp}^{\text{frame-res}}(\Pi_{\text{InSy}}, \mathcal{A}, \lambda)$:
$ispp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda))$
$(pk, st) \leftarrow \mathcal{A}(ispp)$
$(u, dslink) \leftarrow \mathcal{A}^{\mathsf{oracles}}(st)$
where $\mathsf{oracles} = (\mathbf{Keygen}(), \mathbf{Join}(\cdot, pk),$
$\mathbf{Earn}(\cdot,\cdot), \mathbf{Spend}(\cdot,\cdot))$
Output 1 if
$\mathsf{VrfyDs}(ispp, dslink, upk_u) = 1$
return 0

Figure 3: Experiments defining anonymity and framing resistance for InSy

updates $token_u \leftarrow token^*$, $dsid_u \leftarrow dsid^*$ and $v_u \leftarrow v_u - k$. If $token^* = \perp$, the oracle outputs $\perp$ and all further calls to any oracles concerning u are ignored.[1]

Honest Issuer. To model an honest issuer, we define the following oracles:

- **IssuerKeyGen**() runs $(pk, sk) \leftarrow \mathsf{IssuerKeyGen}(ispp)$. It stores pk and sk for further use. It initially sets the set of users $\mathcal{U} \leftarrow \emptyset$ and sets the double-spend database $\mathcal{DB}$ to the empty graph. Furthermore, initially $v_{\text{earned}}, v_{\text{spent}} \leftarrow 0$. Further calls to this oracle are ignored. This oracle must be called before any of the other issuer-related oracles. The oracle outputs pk.
- **Issue**(upk) if $upk \in \mathcal{U}$, the request is ignored. Otherwise, the oracle runs $\mathsf{Issue}(ispp, pk, upk, sk)$ and adds upk to $\mathcal{U}$.
- **Credit**(k) for $k \in \mathbb{N}$, runs $\mathsf{Credit}(ispp, pk, k, sk)$ and sets $v_{\text{earned}} \leftarrow v_{\text{earned}} + k$.
- **Deduct**(k) for $k \in \mathbb{N}$, waits to receive $dsid$. It then runs $\mathsf{Deduct}(ispp, pk, k, dsid, sk) \rightarrow (b, dstag)$. If $b = 0$, it outputs $\perp$. Otherwise, it chooses a fresh spend handle s and stores $(dsid_s, dstag_s, k_s) \leftarrow (dsid, dstag, k)$. Then it outputs s and increments $v_{\text{spent}} \leftarrow v_{\text{spent}} + k$.
- **DBsync**(s) runs $\mathcal{DB}' \leftarrow \mathsf{DBsync}((dsid_s, dstag_s, k_s), \mathcal{DB})$. It then updates $\mathcal{DB} \leftarrow \mathcal{DB}'$ and recomputes v_{invalid} as the sum of values k associated with invalid transactions within $\mathcal{DB}'$.

We now define security for incentive systems. Anonymity guarantees that honest users running Spend and Earn are indistinguishable.

Definition 11 (Anonymity)**.** We define the experiment $\mathrm{Exp}^{\text{anon-X}}$ in Fig. 3 for $X \in \{\mathsf{Earn}, \mathsf{Spend}\}$. We say that incentive system Π_{InSy} is *anonymous* if for both $X \in \{\mathsf{Earn}, \mathsf{Spend}\}$ and for all ppt $\mathcal{A}$ it holds that $|\Pr[\mathrm{Exp}_0^{\text{anon-X}}(\Pi, \mathcal{A}, \lambda) = 1] - \Pr[\mathrm{Exp}_1^{\text{anon-X}}(\Pi, \mathcal{A}, \lambda) = 1]| \leq negl(\lambda)$ for all λ. $\diamond$

An incentive system has framing resistance if honest users cannot be falsely accused of double spending by a dishonest issuer.

Definition 12 (Framing resistance)**.** We define experiment $\mathrm{Exp}^{\text{frame-res}}$ in *Fig.* 3. We say that incentive system Π_{InSy} is *framing resistant* if for all ppt $\mathcal{A}$, there exists a negligible function *negl* s.t. $\Pr[\mathrm{Exp}^{\text{frame-res}}(\Pi, \mathcal{A}, \lambda) = 1] \leq negl(\lambda)$ for all λ. $\diamond$

Soundness should guarantee that users cannot spend more points than they have earned (excluding spend operations detected as double-spending).

[1] Spending the same *token* twice would be considered double-spending, even if one of the Spend operations fails. Hence after a failed Spend operation, the user must not attempt to use her old *token*.

$\mathrm{Exp}^{\mathrm{sound}}(\Pi_{\mathrm{InSy}}, \mathcal{A}, \lambda)$:
$ispp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda))$
Call $pk \leftarrow \mathbf{IssuerKeyGen}()$
$(\mathsf{halt}) \leftarrow \mathcal{A}^{\mathbf{Issue}(\cdot),\mathbf{Credit}(\cdot),\mathbf{Deduct}(\cdot),\mathbf{DBsync}(\cdot)}(ispp, pk)$
If $v_{\mathrm{spent}} - v_{\mathrm{invalid}} > v_{\mathrm{earned}}$ and $\mathcal{A}$ has queried $\mathbf{DBsync}(s)$ for all spending record handles s output by the **Deduct** oracle, return 1
If $\mathcal{DB}$ contains some $(upk, dslink)$ associated with some $dsid$ such that $\mathsf{VrfyDs}(ispp, dslink, upk) \neq 1$ or $upk \notin \mathcal{U}$, return 1
return 0

Figure 4: Soundness experiment for InSy

Definition 13 (Soundness)**.** We define the experiment $\mathrm{Exp}^{\mathrm{sound}}$ in *Fig.* 4. We say that incentive system Π_{InSy} is *sound* if for all ppt $\mathcal{A}$, there exists a negligible function *negl* with $\Pr[\mathrm{Exp}^{\mathrm{sound}}(\Pi, \mathcal{A}, \lambda) = 1] \leq negl(\lambda)$ for all λ. ◇

6 Construction of an Incentive System from UACS

We construct an incentive system as follows: Users hold UACS credentials with attributes $(usk, dsid, dsrnd, v)$ as explained in the introduction. We ensure that *dsid*s are random by choosing new *dsid*s as follows: The user commits to a random $dsid_{\mathrm{usr}} \leftarrow \mathbb{Z}_p$ with an additively malleable commitment scheme, then the issuer chooses random $dsid_{\mathrm{isr}} \leftarrow \mathbb{Z}_p$ and sends it to the user. Then both change the commitment to $dsid = dsid_{\mathrm{usr}} + dsid_{\mathrm{isr}}$ using additive malleability and embed *dsid* in the credential. If the user or the issuer is honest, *dsid* is uniformly random, which factors into anonymity or soundness, respectively. The rest of the construction follows its description in the introduction.

Construction 14. Let $\Pi_{\mathcal{C}}$ be an UACS, $\Pi_{\mathcal{E}}$ be a public-key encryption scheme, and let $\Pi_{\mathcal{C}^+}$ be an additively malleable commitment scheme. We define the incentive system Π_{InSy} as follows:

$\mathsf{Setup}(pp) \to ispp$ runs $cpp \leftarrow \mathsf{Setup}_{\mathcal{C}}(pp)$. pp fixes an attribute space $\mathbb{A}$ and message space $\mathcal{M}_{\mathcal{E}}$ for the encryption scheme. We assume $\mathbb{A} = \mathbb{Z}_p$ for some super-poly p and set $v_{max} = p - 1$. Setup chooses a commitment key $pk_{\mathcal{C}^+} \leftarrow \mathsf{KeyGen}_{\mathcal{C}^+}(pp)$. It outputs $ispp = (pp, cpp, pk_{\mathcal{C}^+})$.

$\mathsf{KeyGen}(ispp) \to (upk, usk)$ generates an encryption key pair $usk_{\mathcal{E}} \leftarrow \mathsf{KeyGen}_{\mathcal{E}}(pp)$ and $upk_{\mathcal{E}} = \mathsf{ComputePK}_{\mathcal{E}}(pp, usk_{\mathcal{E}})$. It outputs $upk = upk_{\mathcal{E}}$ and $usk = usk_{\mathcal{E}}$.

$\mathsf{IssuerKeyGen}(ispp) \to (pk, sk)$ generates and outputs a credential issuer key pair $(pk, sk) \leftarrow \mathsf{IssuerKeyGen}_{\mathcal{C}}(cpp, 1^n)$ for $n = 4$.

$\mathsf{Issue}(ispp, pk, upk, sk) \leftrightarrow \mathsf{Join}(ispp, pk, upk, usk) \to (token, dsid)$ the receiver picks $dsid_{\mathrm{usr}} \leftarrow \mathbb{Z}_p$ and computes $(C_{\mathrm{usr}}, open) \leftarrow \mathsf{Commit}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, dsid_{\mathrm{usr}})$. The issuer replies with $dsid_{\mathrm{isr}} \leftarrow \mathbb{Z}_p$. Both parties compute $C_{\mathrm{dsid}} = \mathsf{Add}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, C_{\mathrm{usr}}, dsid_{\mathrm{isr}})$. Then the receiver sets $dsid = dsid_{\mathrm{usr}} + dsid_{\mathrm{isr}}$, chooses $dsrnd \leftarrow \mathbb{Z}_p$, and sets $\alpha = (usk_{\mathcal{E}}, dsid, dsrnd, open)$. Then the issuer runs $\mathsf{Issue}_{\mathcal{C}}(cpp, pk, \psi, sk)$ and the receiver runs $\mathsf{Receive}_{\mathcal{C}}(cpp, pk, \psi, \alpha) \to cred$. Here, the update function is set to $\psi(\bot, (usk, dsid, dsrnd, open)) = (usk, dsid, dsrnd, 0)$, if user public key $upk = \mathsf{ComputePK}_{\mathcal{E}}(pp, usk)$ and it holds that $\mathsf{Vrfy}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, C_{\mathrm{dsid}}, dsid, open) = 1$. Otherwise, $\psi(\bot, \alpha) = \bot$. If $cred \neq \bot$, the receiver outputs $token = (dsid, dsrnd, v = 0, cred)$ and $dsid$. Otherwise, the receiver outputs $\bot$.

$\mathsf{Credit}(ispp, pk, k, sk) \leftrightarrow \mathsf{Earn}(ispp, pk, k, usk, token) \to token^*$ checks that $v + k \leq v_{max}$ for the $token = (dsid, dsrnd, v, cred)$. It then works as follows: The issuer runs $\mathsf{Update}_{\mathcal{C}}(cpp, pk, \psi, sk)$ and the receiver $\mathsf{UpdRcv}_{\mathcal{C}}(cpp, pk, \psi, \alpha, cred) \to cred^*$ with $\alpha = \bot$. Here, the update function is set to $\psi((usk_{\mathcal{E}}, dsid, dsrnd, v), \cdot) = (usk_{\mathcal{E}}, dsid, dsrnd, v + k)$. If $cred^* \neq \bot$, the user outputs $token^* = (dsid, dsrnd, v + k, cred^*)$.

$(token^*, dsid^*) \leftarrow \mathsf{Spend}(ispp, pk, k, dsid, usk, token) \leftrightarrow \mathsf{Deduct}(ispp, pk, k, dsid, sk) \rightarrow (b, dstag)$ for $token = (dsid, dsrnd, v, cred)$ checks that $v \geq k$. Then:

- The user chooses random $dsid^*_{\text{usr}} \leftarrow \mathbb{Z}_p$ and generates $({C_{\text{usr}}}^*, open^*) \leftarrow \mathsf{Commit}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, dsid^*_{\text{usr}})$. He sends ${C_{\text{usr}}}^*$ to the issuer.
- The issuer chooses a random challenge $\gamma \leftarrow \mathbb{Z}_p$ and a random $dsid^*_{\text{isr}} \leftarrow \mathbb{Z}_p$, and sends both to the user.
- Issuer and user each compute ${C_{\text{dsid}}}^* = \mathsf{Add}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, {C_{\text{usr}}}^*, dsid^*_{\text{isr}})$.
- The user prepares new values $dsid^* = dsid^*_{\text{usr}} + dsid^*_{\text{isr}}$ and $dsrnd^* \leftarrow \mathbb{Z}_p$ for his next token and sets $\alpha = (dsid^*, dsrnd^*, open^*)$.
- The user computes $c = usk_{\mathcal{E}} \cdot \gamma + dsrnd$.
- The user encrypts $dsid^*$ as $ctrace \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(pp, upk_{\mathcal{E}}, dsid^*)$.
- The user sends c, $ctrace$ to the issuer.
- The issuer runs $b \leftarrow \mathsf{Update}_{\mathcal{C}}(cpp, pk, \psi, sk)$ and the user runs $cred^* \leftarrow \mathsf{UpdRcv}_{\mathcal{C}}(cpp, pk, \psi, \alpha, cred)$. Here, $\psi((usk_{\mathcal{E}}, dsid, dsrnd, v), (dsid^*, dsrnd^*, open^*)) = (usk_{\mathcal{E}}, dsid^*, dsrnd^*, v-k)$
  - $dsid$ is the same as in the Deduct input,
  - $v \geq k$,
  - $\mathsf{Vrfy}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, {C_{\text{dsid}}}^*, dsid^*, open^*)$,
  - $c = usk_{\mathcal{E}} \cdot \gamma + dsrnd$, and
  - $\mathsf{Decrypt}_{\mathcal{E}}(pp, usk_{\mathcal{E}}, ctrace) = dsid^*$.

  Otherwise, $\psi(\dots) = \perp$.
- If $cred^* \neq \perp$, the user outputs $(token^* = (dsid^*, dsrnd^*, v-k, cred^*), dsid^*)$.
- The issuer outputs b and, if $b = 1$, $dstag = (c, \gamma, ctrace)$.

$\mathsf{Link}(ispp, dstag, dstag') \rightarrow (upk, dslink)$ given $dstag = (c, \gamma, ctrace)$, $dstag' = (c', \gamma', ctrace')$, outputs $dslink = (c-c')/(\gamma-\gamma')$ (the intent is that $dslink = usk$) and $upk = \mathsf{ComputePK}(pp, dslink)$.

$\mathsf{Trace}(ispp, dslink, dstag) \rightarrow dsid^*$ for $dstag = (c, \gamma, ctrace)$ retrieves $dsid^*$ by decrypting $ctrace$ as follows $\mathsf{Decrypt}_{\mathcal{E}}(pp, dslink, ctrace) = dsid^*$.

$\mathsf{VrfyDs}(ispp, dslink, upk) \rightarrow b$ outputs 1 iff $\mathsf{ComputePK}(pp, dslink) = upk$.

It is easy to check correctness given that $dsid$s are chosen randomly from $\mathbb{Z}_p$.

Theorem 15. If $\Pi_{\mathcal{C}}$ has simulation anonymity (Definition 5), $\Pi_{\mathcal{E}}$ is key-ind. CPA secure (Definition 20), $\Pi_{\mathcal{C}^+}$ is computational hiding, then the scheme Π_{InSy} (Construction 14) guarantees *anonymity* (Definition 11).

Proof sketch. The adversary $\mathcal{A}$ is asked to distinguish if it talks to user u_0 or u_1 in the challenge phase. Both users are determined by $\mathcal{A}$. We will first handle the easy case of $\text{Exp}_b^{\text{anon-X}}(\Pi_{\text{InSy}}, \mathcal{A}, \lambda)$ for $X = \mathsf{Earn}$: everything that the adversary $\mathcal{A}$ sees perfectly hides the user's secret usk and $dsid$. For the case $X = \mathsf{Spend}$ and user u_b, let i be the spend operation in the challenge phase and $i-1$ the previous spend operation in the setup phase. During spend $i-1$, the adversary $\mathcal{A}$ gets $\mathsf{Encrypt}_{\mathcal{E}}(pp, upk_b, dsid_i)$ and can compute $\mathsf{Commit}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, dsid_i)$ from the commitment to $dsid_{\text{usr}}$ that he receives. In spend i, $\mathcal{A}$ gets (1) $\mathsf{Encrypt}_{\mathcal{E}}(pp, upk_b, dsid_{i+1})$, (2) $\mathsf{Commit}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, dsid_{i+1})$, and (3) $dsid_i$. For (2), observe that $\mathsf{Commit}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, dsid_{i+1})$ has no influence on $\mathcal{A}$'s advantage since it is independent of b. If we look at (1), we observe that the encryption is generated under upk_b. Therefore, in addition to CPA security, we need that the keys of the users are indistinguishable. Considering (3), observe that the commitment to

$dsid_i$ (in spend $i-1$) is computationally hiding. Furthermore, to link $\mathsf{Encrypt}_{\mathcal{E}}(pp, upk_b, dsid_i)$ or $\mathsf{Commit}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, dsid_i)$ from spend $i-1$ to $dsid_i$ revealed in spend i, $\mathcal{A}$ has to break (key-ind.) CPA security of $\Pi_{\mathcal{E}}$ or comp. hiding of $\Pi_{\mathcal{C}^+}$.

An extended formal proof is given in Appendix D.2.

Theorem 16. If $\Pi_{\mathcal{C}}$ is sound (Definition 6), $\mathsf{ComputePK}_{\mathcal{E}}(pp, \cdot)$ is injective, and $\Pi_{\mathcal{C}^+}$ is perfectly binding (Definition 21), then Π_{InSy} (Construction 14) is *sound* (Definition 13).

Proof sketch. The proof is a reduction to soundness of the underlying updatable credential system. Let $\mathcal{A}$ be an attacker against incentive system soundness. We construct $\mathcal{B}$. $\mathcal{B}$ simulates $\mathcal{A}$'s view perfectly by replacing $\mathsf{Issue}_{\mathcal{C}}$ and $\mathsf{Update}_{\mathcal{C}}$ calls with calls to the corresponding UACS oracles. Let error be the event that (1) $\mathcal{B}$ has output the same challenge δ in two different **Deduct** runs, or (2) there were two commitments $C_{\text{dsid}}, C_{\text{dsid}}{}'$ in two runs of **Deduct** or **Issue** such that the commitments can be opened to two different messages. (1) happens with negligible probability ($\delta \leftarrow \mathbb{Z}_p$), so does (2) because $dsid_{\text{isr}}, dsid^*_{\text{isr}} \leftarrow \mathbb{Z}_p$ and the commitment scheme $\Pi_{\mathcal{C}^+}$ is perfectly binding. It then remains to show that if error does not happen and there exists a consistent explanation list $\mathcal{L}$, then $\mathcal{A}$ cannot wins (implying that unless error, if $\mathcal{A}$ wins, then $\mathcal{B}$ wins as there is no $\mathcal{L}$ that would make it lose). The proof of this is somewhat technical, but essentially, we look at each user individually. For this user, there exists some "canonical" sequence of spend and earn operations in $\mathcal{L}$ that does not involve any spend operations marked as invalid (in the double-spending database $\mathcal{DB}$). From the design of update functions and consistency of $\mathcal{L}$, it is clear that in such a sequence, the value accumulated by earn operations cannot be smaller than the value spent through spend operations, i.e. the desired property $v_{\text{earned}} \geq v_{\text{spent}} - v_{\text{invalid}}$ holds if we only consider these canonical operations. The rest of the proof deals with ensuring that spend operations that are not part of the canonical sequence have all been marked invalid in $\mathcal{DB}$ (such that removing all non-canonical operations from consideration does not change $v_{\text{spent}} - v_{\text{invalid}}$ and only decreases v_{earned}). Because of $\neg$error, challenges γ do not repeat and any two attribute-vectors that share the same $dsid$ have the same $usk, dsrnd$. This implies that extracting usk from two transactions with the same $dsid$ works without error (given $c = usk \cdot \gamma + dsrnd$ and the definition of Link). Since any extracted usk is correct in this sense, the tracing of $dsid$ as in DBsync works as intended, i.e. all invalid transactions will be marked as such in $\mathcal{DB}$ as required.

The full proof can be found in Appendix D.3.

Theorem 17. If $\Pi_{\mathcal{E}}$ is CPA-secure and $\Pi_{\mathcal{C}}$ has simulation anonymity, then Π_{InSy} (Construction 14) is framing resistant.

Framing resistance follows easily via reduction to $\Pi_{\mathcal{E}}$'s (key-ind.) CPA security: An adversary who can frame an honest user needs to be able to compute the secret key usk for the user's $upk = \mathsf{ComputePK}_{\mathcal{E}}(pp, usk)$.

7 Instantiation and Performance of our Incentive System

We instantiated Construction 14 using the signature scheme by Pointcheval and Sanders [PS16] for the UACS, and ElGamal as the public-key encryption scheme and malleable commitment. Using the open-source Java library `upb.crypto` and the bilinear group (bn256) provided by `mcl`[2] we implemented this instantiation and ran it on a phone (typical user device) and a laptop (approximate issuer device). In Table 1 we focus on the execution time (in ms) of the protocols, excluding communication cost.

[2] upb.crypto: `https://github.com/upbcuk`. mcl: `https://github.com/herumi/mcl`

Table 1: Avg. performance of our implementation over 100 runs in milliseconds. Emphasized: typical execution platform for each algorithm.

Device	Issue	Join	Credit	Earn	Deduct	Spend
Google Pixel (Phone, Snapdragon 821)	56	**76**	122	**110**	353	**390**
Surface Book 2 (Laptop, i7-8650U)	**10**	13	**17**	18	**64**	69

References

[Ame19] American Express Company. American express membership rewards. `https://global.americanexpress.com/rewards`, January 2019.

[BB18] Johannes Blömer and Jan Bobolz. Delegatable attribute-based anonymous credentials from dynamically malleable signatures. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 221–239. Springer, Heidelberg, July 2018.

[BBB+18] Kai Bemmann, Johannes Blömer, Jan Bobolz, Henrik Bröcher, Denis Diemert, Fabian Eidens, Lukas Eilers, Jan Haltermann, Jakob Juhnke, Burhan Otour, Laurens Porzenheim, Simon Pukrop, Erik Schilling, Michael Schlichtig, and Marcel Stienemeier. Fully-featured anonymous credentials with reputation system. In *ARES'18*. ACM, 2018.

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 356–374. Springer, Heidelberg, March 2008.

[CCs08] Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 234–252. Springer, Heidelberg, December 2008.

[CDD17] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 683–699. ACM Press, October / November 2017.

[CGH11] Scott E. Coull, Matthew Green, and Susan Hohenberger. Access controls for oblivious and anonymous systems. *ACM Trans. Inf. Syst. Secur.*, 2011.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 302–321. Springer, Heidelberg, May 2005.

[CKS10] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. Solving revocation with efficient update of anonymous credentials. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 454–471. Springer, Heidelberg, September 2010.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 418–430. Springer, Heidelberg, May 2000.

[DMM+18] Dominic Deuber, Matteo Maffei, Giulio Malavolta, Max Rabkin, Dominique Schröder, and Mark Simkin. Functional credentials. *PoPETs*, 2018, 2018.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

[HHNR17] Gunnar Hartung, Max Hoffmann, Matthias Nagel, and Andy Rupp. BBA+: improving the security and applicability of privacy-preserving point collection. In *CCS*. ACM, 2017.

[JR16] Tibor Jager and Andy Rupp. Black-box accumulation: Collecting incentives in a privacy-preserving way. *PoPETs*, 2016, 2016.

[MDPD15] Milica Milutinovic, Italo Dacosta, Andreas Put, and Bart De Decker. uCentive: An efficient, anonymous and unlinkable incentives scheme. In *TrustCom*. IEEE, 2015.

[PAY19] PAYBACK GmbH. Payback. `https://www.payback.net/`, January 2019.

[PS16] David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, February / March 2016.

A Security Definitions for Building Blocks

Definition 18 (Unforgeability)**.** Consider the following unforgeability game $\mathrm{Exp}^{\text{blind-uf}}(\Pi, \mathcal{A}, \lambda)$ for a blind signature scheme Π:

- The experiment runs $pp \leftarrow \mathcal{G}(1^\lambda)$ and hands pp to $\mathcal{A}$. $\mathcal{A}$ responds with 1^n for some $n \in \mathbb{N}$. The experiment runs $(pk, sk) \leftarrow \mathsf{KeyGen}(pp, 1^n)$ and hands pk to $\mathcal{A}$.
- $\mathcal{A}$ can query signatures by announcing c, $\vec{m} \in \mathcal{M}^n$ and r such that $c = \mathsf{Commit}(pp, pk, \vec{m}, r)$. The experiment runs $\mathsf{BlindSign}(pp, pk, sk, c)$ interacting with $\mathcal{A}$ and records $\vec{m}$.
- Eventually, $\mathcal{A}$ outputs $\vec{m}^*$ and σ^*. The experiment outputs 1 iff $\mathsf{Vrfy}(pp, pk, \vec{m}^*, \sigma^*) = 1$ and $\vec{m}^*$ was not recorded in any query.

Π has *blind unforgeability* if for all ppt $\mathcal{A}$ there exists *negl* such that $\Pr[\mathrm{Exp}^{\text{blind-uf}}(\Pi, \mathcal{A}, \lambda) = 1] \leq \mathit{negl}(\lambda)$ for all λ. ◇

Definition 19 (Perfect msg privacy)**.** A blind signature scheme has perfect message privacy if

- *"the commitment scheme is perfectly hiding"*: For all $\vec{m}_0, \vec{m}_1 \in M^n$, $\mathsf{Commit}(pp, pk, \vec{m}_0, r_0)$ is distributed the same as $\mathsf{Commit}(pp, pk, \vec{m}_1, r_1)$ over the random choice of r_0, r_1.
- *"*$\mathsf{BlindRcv}$ *does not reveal the message"*: for any two messages $\vec{m}_0, \vec{m}_1 \in M^n$ and all (unrestricted) $\mathcal{A}$:

$$\begin{aligned}&(\mathrm{output}_{\mathcal{A}}[\mathcal{A}(C_0) \leftrightarrow \mathsf{BlindRcv}(pp, pk, \vec{m}_0, r_0)], \chi_0)\\ \approx\ &(\mathrm{output}_{\mathcal{A}}[\mathcal{A}(C_1) \leftrightarrow \mathsf{BlindRcv}(pp, pk, \vec{m}_1, r_1)], \chi_1)\end{aligned}$$

  where r_0, r_1 is chosen uniformly at random, $C_j = \mathsf{Commit}(pp, pk, \vec{m}_j, r_j)$ and χ_j is an indicator variable with $\chi_j = 1$ if and only if $\mathsf{Vrfy}(pp, pk, \vec{m}_j, \sigma_j) = 1$ for the local output σ_j of $\mathsf{BlindRcv}$ in either case. ◇

While this definition may seem strong, it is satisfied, for example, by the Pointcheval Sanders blind signature scheme [PS16], where Commit is a effectively a (perfectly hiding) Pedersen commitment, Their $\mathsf{BlindRcv}$ (in our formulation without zero-knowledge proof) does not send any messages (meaning the output of $\mathcal{A}$ is clearly independent of $\vec{m}$), and the χ_j bit (validity of the resulting signature) is also independent of the committed message.

Definition 20 (Key-indistinguishable CPA)**.** Let $\Pi_{\mathcal{E}}$ be a public-key encryption scheme. Consider the following experiment $\mathrm{Exp}_b^{\text{key-ind}}(\Pi_{\mathcal{E}}, \mathcal{A}, \lambda)$ for $b \in \{0, 1\}$.

- The experiment generates $pp \leftarrow \mathcal{G}(1^\lambda)$ and two keys $\mathsf{KeyGen}_{\mathcal{E}}(pp) \to sk_0, sk_1$, hands $\mathcal{A}$ the pp and public keys $(pk_0, pk_1) = (\mathsf{ComputePK}_{\mathcal{E}}(pp, sk_0), \mathsf{ComputePK}_{\mathcal{E}}(pp, sk_1))$.
- $\mathcal{A}$ outputs two messages $m_0, m_1 \in M_{pp}$.
- $\mathcal{A}$ gets $\mathsf{Encrypt}_{\mathcal{E}}(pp, \underline{pk_b}, m_b)$ from the experiment and outputs a bit $\hat{b}$.

We say that $\Pi_{\mathcal{E}}$ is key-ind. CPA secure if for all ppt $\mathcal{A}$, there exists a negligible function *negl* s.t.

$$|\Pr[\mathrm{Exp}_0^{\text{key-ind}}(\Pi_{\mathcal{E}}, \mathcal{A}, \lambda) = 1] - \Pr[\mathrm{Exp}_1^{\text{key-ind}}(\Pi_{\mathcal{E}}, \mathcal{A}, \lambda) = 1]| \leq \mathit{negl}(\lambda)$$

Definition 21 (Perfectly binding commitment)**.** A (malleable) commitment scheme is *perfectly binding* if for all $pp \in [\mathcal{G}(1^\lambda)]$, $pk \in [\mathsf{KeyGen}(pp)]$ and all $(c, o) \in [\mathsf{Commit}(pp, pk, m)]$, there exists no m', o' such that $m' \neq m$ and $\mathsf{Vrfy}(pp, pk, c, o', m') = 1$.

Definition 22 (Comp. hiding commitment)**.** Let $\Pi_{\mathcal{C}^+}$ be a malleable commitment scheme. Consider the following experiment $\mathrm{Exp}_b^{\text{hiding}}(\Pi_{\mathcal{C}^+}, \mathcal{A}, \lambda)$.

- $pp \leftarrow \mathcal{G}(1^\lambda)$, $pk \leftarrow \mathsf{KeyGen}(pp)$, $(m_0, m_1, st) \leftarrow \mathcal{A}(pp, pk)$, $m_0, m_1 \in M_{pp}$
- $\hat{b} \leftarrow \mathcal{A}(c, st)$ where $c = \mathsf{Commit}(pp, pk, m_b)$

We say that $\Pi_{\mathcal{C}^+}$ is comp. hiding if for all ppt $\mathcal{A}$, there exists a negligible function *negl* s.t.

$$|\Pr[\mathrm{Exp}_0^{\text{hiding}}(\Pi_{\mathcal{C}^+}, \mathcal{A}, \lambda) = 1] - \Pr[\mathrm{Exp}_1^{\text{hiding}}(\Pi_{\mathcal{C}^+}, \mathcal{A}, \lambda) = 1]| \leq negl(\lambda)$$

B Security Proof for Updatable Credentials

In this section, we sketch the security proofs for Construction 7.

Theorem 8: Anonymity. We define the simulators as follows:

- $\mathfrak{S}_{\mathsf{Setup}}(pp)$ runs the trapdoor generator of the ZKAK and outputs *cpp* (which contains *pp* and the ZKAK common reference string from the trapdoor generator), and the simulation trapdoor *td*.
- $\mathfrak{S}_{\mathsf{Receive}}(td, pk, \psi)$ and $\mathfrak{S}_{\mathsf{UpdRcv}}(td, pk, \psi)$ work very similarly to one another: they both commit to $\vec{0}$ as $c = \mathsf{Commit}_{\mathcal{S}}(pp, pk, \vec{0}, r)$ with random r and send c to $\mathcal{A}$. They then simulate the ZKAK proof (in Receive or UpdRcv) using *td*. Finally, they runs $\mathsf{BlindRcv}_{\mathcal{S}}(pp, pk, \vec{0}, r) \to \sigma$ and compute the bit $b = \mathsf{Vrfy}_{\mathcal{S}}(pp, pk, \vec{0}, \sigma)$. They send b to $\mathcal{A}$.
- $\mathfrak{S}_{\mathsf{ShowPrv}}(td, pk, \phi)$ simulates the ZKAK.

Given that $\Pi_{\mathcal{S}}$ has perfect message privacy by assumption, the commitment c and the bit b computed by the simulator have the same distribution as in Receive or UpdRcv. Simulation of the zero-knowledge arguments produces the correct view for $\mathcal{A}$ by assumption. □

Theorem 9: Soundness. We define the algorithm $\mathcal{E}$ that is supposed to extract an explanation list $\mathcal{L}$ as follows:

- On input $(cpp, r_{\mathcal{A}}, r_{\mathsf{Issue}}, r_{\mathsf{Update}})$, the extractor $\mathcal{E}^{\mathcal{A}}$ first runs $\mathcal{A}$ with randomness $r_{\mathcal{A}}$ and *cpp* until $\mathcal{A}$ halts.
- For the ith query to Issue, Update, or ShowVrfy in this run, $\mathcal{E}$ does the following:
  - if it is a query to Issue and the proof of knowledge within Issue is accepting, then $\mathcal{E}$ uses the proof of knowledge extractor to obtain a witness (α, r). It stores $\alpha_i := \alpha$ on $\mathcal{L}$. If the proof of knowledge is not accepting, it stores some arbitrary $\alpha_i \in \{0,1\}^*$ on $\mathcal{L}$.
  - if it is a query to Update and the proof of knowledge within Update is accepting, then $\mathcal{E}$ uses the proof of knowledge extractor to obtain a witness $(\vec{A}, \sigma, \alpha, r)$. It stores $(\vec{A}_i, \alpha_i) := (\vec{A}, \alpha)$ on $\mathcal{L}$.
  - if it is a query to ShowVrfy and the proof of knowledge within ShowVrfy is accepting, then $\mathcal{E}$ uses the proof of knowledge extractor to obtain a witness $(\vec{A}, \sigma)$. It stores $\vec{A}_i := \vec{A}$ on $\mathcal{L}$.
- $\mathcal{E}$ outputs $\mathcal{L}$.

Since the argument of knowledge extractor runs in expected polynomial time, $\mathcal{E}$ runs in expected polynomial time, too (probability over $r_{\mathcal{A}}$ and $\mathcal{E}$'s random coins).

With this $\mathcal{E}$, the soundness of our updatable credential construction can be reduced to unforgeability of the underlying blind signature scheme $\mathcal{S}$ (Definition 18). Let $\mathcal{E}$ be as above. Let $\mathcal{A}$ be an attacker against $\mathrm{Exp}^{\text{sound}}$. We construct $\mathcal{B}$ against $\mathrm{Exp}^{\text{blind-uf}}$:

- $\mathcal{B}$ runs $\mathcal{A}$ with randomness $r_{\mathcal{A}}$
- $\mathcal{B}$ receives pp from the unforgeability experiment. $\mathcal{B}$ generates cpp from pp and hands cpp to $\mathcal{A}$. $\mathcal{A}$ responds with 1^n for some $n \in \mathbb{N}$. $\mathcal{B}$ hands 1^n to its challenger, receiving pk. $\mathcal{B}$ hands pk to $\mathcal{A}$.
- Whenever $\mathcal{A}$ queries Issue with update function ψ, $\mathcal{B}$ checks the proof of knowledge. If it accepts, $\mathcal{B}$ uses the proof of knowledge extractor to obtain a witness (α, r). $\mathcal{B}$ submits $\vec{m} := \psi(\perp, \alpha)$, r, and $c := \mathsf{Commit}_{\mathcal{S}}(pp, pk, \vec{m}, r)$ to its challenger, who starts running $\mathsf{BlindSign}_{\mathcal{S}}(pp, pksk, c)$. $\mathcal{B}$ relays the messages for $\mathsf{BlindSign}_{\mathcal{S}}$ between its challenger and $\mathcal{A}$.
- Whenever $\mathcal{A}$ queries Update with update function ψ, $\mathcal{B}$ checks the proof of knowledge. If it accepts, $\mathcal{B}$ uses the proof of knowledge extractor to obtain a witness $(\vec{A}, \sigma, \alpha, r)$. If $\mathcal{B}$ has not queried its challenger for $\vec{A}$ before, it outputs $\vec{m} := \vec{A}$ and σ as a forgery. Otherwise, $\mathcal{B}$ submits $\vec{m} := \psi(\vec{A}, \alpha)$, r, and $c := \mathsf{Commit}_{\mathcal{S}}(pp, pk, \vec{m}, r)$ to its challenger, who starts running $\mathsf{BlindSign}_{\mathcal{S}}(pp, pksk, c)$. $\mathcal{B}$ relays the messages for $\mathsf{BlindSign}\mathcal{S}$ between its challenger and $\mathcal{A}$.
- Whenever $\mathcal{A}$ queries $\mathsf{ShowVrfy}$ with predicate ϕ, $\mathcal{B}$ checks the proof of knowledge. If it accepts, $\mathcal{B}$ uses the proof of knowledge extractor to obtain a witness $(\vec{A}, \sigma)$. If $\mathcal{B}$ has not queried its challenger for $\vec{A}$ before, it outputs $\vec{m} := \vec{A}$ and σ as a forgery.
- Eventually, $\mathcal{A}$ and halts. $\mathcal{B}$ runs $\mathcal{E}^{\mathcal{A}}(cpp, r_{\mathcal{A}}, r_{\mathsf{Issue}}, r_{\mathsf{Update}})$ (using the same random coins for $\mathcal{E}$ that $\mathcal{B}$ used for its extraction of proofs of knowledge, ensuring that the output of $\mathcal{E}$ will be consistent with the values extracted by $\mathcal{B}$ before) to obtain $\mathcal{L}$.
- Then $\mathcal{B}$ halts.

Analysis: Whenever $\mathcal{B}$ outputs a signature forgery, it is guaranteed that the signature is valid (since they are valid witnesses in a proof of knowledge for a relation that requires signature validity). If $\mathcal{B}$ outputs a forgery during an Update or $\mathsf{ShowVrfy}$ query, by construction it has never asked for the message to be signed before.

It is easy to see that the simulation is perfect. If $\mathcal{B}$ does not halt before $\mathcal{A}$ halts, the output $\mathcal{L}$ of $\mathcal{E}$ necessarily fulfills argument consistency: Suppose for contradiction that $\mathcal{L}$ is not consistent, i.e. there is some index i such that $\mathcal{L}$ is inconsistent for that index. Let E_i be as prescribed in the soundness experiment given $\mathcal{L}$. Note that before the ith query, $\mathcal{B}$ has only queried its oracle for signatures on messages $\vec{A} \in E_{i-1}$.

- Assume i belongs to an Issue query. By definition i cannot have caused the inconsistency.
- Assume i belongs to an Update query with update function ψ_i. Then the entry on $\mathcal{L}$ is some $(\vec{A}_i, \alpha_i)$. Because i caused the inconsistency, Update has output 1 (implying that $\mathcal{B}$ runs the proof of knowledge extractor and obtained the witness $(\vec{A}, \sigma, \alpha, r)$) and (1) $\psi_i(\vec{A}_i, \alpha_i) = \perp$ or (2) $\vec{A}_i \notin E_{i-1}$. (1) can be ruled out since $\psi_i(\vec{A}_i, \alpha_i) \neq \perp$ is guaranteed by the proof of knowledge statement and hence by its extractor. If (2) happens, then $\mathcal{B}$ halts and claims a forgery (as it has not queried $\vec{A}_i$ to its oracle before), contradicting that $\mathcal{B}$ does not halt before $\mathcal{A}$ halts.
- Assume i belongs to a $\mathsf{ShowVrfy}$ query with predicate ϕ_i. This case is handled analogously to Update.

So we know that if $\mathcal{B}$ does not halt before $\mathcal{A}$ halts, then $\mathcal{E}$ outputs a consistent $\mathcal{L}$, implying that $\Pr[\mathrm{Exp}^{\text{blind-uf}}(\Pi_{\mathcal{S}}, \mathcal{B}, \lambda)] \geq \Pr[\mathrm{Exp}^{\text{sound}}(\Pi, \mathcal{A}, \mathcal{E}, \lambda) = 1]$. So if for $\mathcal{E}$ as defined above, there exists an adversary $\mathcal{A}$ with non-negligible success probability, then there exists $\mathcal{B}$ (as defined above) with non-negligible success probability against the blind signature scheme. By assumption, such a $\mathcal{B}$ does not exist, hence the updatable credential system is sound. (Note that $\mathcal{B}$ runs in expected polynomial time. This can be converted to polynomial time by trading off success probability using Markov's inequality.) $\square$

C Concrete Construction from Pointcheval Sanders Blind Signatures

We present a concrete construction based on Pointcheval Sanders blind signatures [PS16] for the UACS and ElGamal encryption for the public-key encryption scheme and the additively malleable commitment. For this, we follow the generic construction of UACS (Construction 7) and the incentive system (Construction 14) closely with one change: We sign $dsid \in \mathbb{Z}_p$, but we *encrypt* $Dsid = w^{dsid} \in \mathbb{G}_1$. Hence, when tracing double-spent transactions, one only learns $Dsid^* = w^{dsid^*}$ instead of $dsid^*$. This is not a restriction since the output of Trace is only needed to quickly find the corresponding transaction to $dsid^*$. So in practice, the issuer would store $Dsid$ instead of $dsid$ for every transaction that he observes, and then use $Dsid$ to quickly find the transaction pointed at by Trace. Note that the security of the construction is not impacted (the same security proofs still apply almost verbatim).

Construction 23 (Incentive system from Pointcheval Sanders signatures)**.** Let $(\mathsf{KeyGen}_{\mathrm{PS}}, \mathsf{Commit}_{\mathrm{PS}}, \mathsf{BlindSign}_{\mathrm{PS}}, \mathsf{BlindRcv}_{\mathrm{PS}}, \mathsf{Vrfy}_{\mathrm{PS}})$ denote the Pointcheval Sanders signature scheme [PS16].

$\mathcal{G}(1^\lambda) \to pp$ generates and outputs a type 3 bilinear group $pp = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_{\mathrm{T}}, p, e)$ of prime order p.

$\mathsf{Setup}(pp) \to ispp$ and chooses a random $w \leftarrow \mathbb{G}_1$ as a shared base for ElGamal encryption $\Pi_{\mathcal{E}}$ and $g, h \leftarrow \mathbb{G}_1$ for the malleable commitment $\Pi_{\mathcal{C}^+}$ (also ElGamal). Setup also generates a Pedersen commitment key $g_{\text{Damgård}}, h_{\text{Damgård}} \leftarrow \mathbb{G}_1$ and a collision-resistant hash function H, both for Damgård's technique [Dam00], enabling efficient simulation of ZKAK protocols. We omit these values in the following. It outputs $ispp = (pp, w, g, h)$. The maximum point score is $v_{max} = p - 1$.

$\mathsf{KeyGen}(ispp) \to (upk, usk)$ generates an encryption key pair by choosing a random $usk \leftarrow \mathbb{Z}_p$ and computing $upk = w^{usk}$. It outputs (upk, usk).

$\mathsf{IssuerKeyGen}(ispp) \to (pk, sk)$ generates keys $(pk_{\mathrm{PS}}, sk_{\mathrm{PS}}) \leftarrow \mathsf{KeyGen}_{\mathrm{PS}}(pp, 1^{n=4})$. We write the keys as $sk_{\mathrm{PS}} = (x, y_1, \ldots, y_4)$ and $pk_{\mathrm{PS}} = (g, g^{y_1}, \ldots, g^{y_4}, \tilde{g}, \tilde{g}^x, \tilde{g}^{y_1}, \ldots, \tilde{g}^{y_4})$. $\mathsf{IssuerKeyGen}$ outputs $pk = pk_{\mathrm{PS}}$ and $sk = sk_{\mathrm{PS}}$.

$\mathsf{Issue}(ispp, pk, upk, sk) \leftrightarrow \mathsf{Join}(ispp, pk, upk, usk) \to (token, dsid)$ works as follows:

- The user chooses random $dsid_{\mathrm{usr}} \leftarrow \mathbb{Z}_p$ and computes the commitment $C_{\mathrm{usr}} = (g_{\mathrm{usr}}^{dsid} \cdot h^{open}, g^{open})$ for a random $open \leftarrow \mathbb{Z}_p$. It sends C_{usr} to the issuer.
- The issuer replies with a random $dsid_{\mathrm{isr}} \leftarrow \mathbb{Z}_p$. Both issuer and user compute $C_{\mathrm{dsid}} = (g_{\mathrm{usr}}^{dsid} \cdot h^{open} \cdot g_{\mathrm{isr}}^{dsid}, g^{open})$.
- The user sets $dsid = dsid_{\mathrm{usr}} + dsid_{\mathrm{isr}}$ and chooses random $dsrnd, r \leftarrow \mathbb{Z}_p$, computes $Dsid = w^{dsid}$ and sends $c = (g^{y_1})^{usk} \cdot (g^{y_2})^{dsid} \cdot (g^{y_3})^{dsrnd} \cdot g^r$ to the issuer.
- The user proves $\mathsf{ZKAK}[(usk, dsid, dsrnd, r, open); c = (g^{y_1})^{usk} \cdot (g^{y_2})^{dsid} \cdot (g^{y_3})^{dsrnd} \cdot g^r \wedge upk = w^{usk} \wedge C_{\mathrm{dsid}} = (g^{dsid} \cdot h^{open}, g^{open})]$.
- If the proof is accepted, the issuer sends $\sigma'_{\mathrm{PS}} = (\sigma'_0, \sigma'_1) = (g^{r'}, (c \cdot g^x)^{r'})$ for a random $r' \leftarrow \mathbb{Z}_p^*$ to the user.
- The user unblinds the signature as $\sigma_{\mathrm{PS}} = (\sigma'_0, \sigma'_1 \cdot (\sigma'_0)^{-r})$.
- The user checks $\mathsf{Vrfy}_{\mathrm{PS}}(pp, pk_{\mathrm{PS}}, (usk, dsid, dsrnd, 0), \sigma_{\mathrm{PS}}) \stackrel{!}{=} 1$. If the checks succeed, it outputs $token = (dsid, dsrnd, v = 0, \sigma_{\mathrm{PS}})$ and $Dsid$, otherwise it outputs $\perp$.

$\mathsf{Credit}(ispp, pk, k, sk) \leftrightarrow \mathsf{Earn}(ispp, pk, k, usk, token) \to token^*$ for $token = (dsid, dsrnd, v, \sigma_{\mathrm{PS}} = (\sigma_0, \sigma_1))$ works as follows:

- The user computes randomized signatures $(\sigma'_0, \sigma'_1) = (\sigma_0^r, (\sigma_1 \cdot \sigma_0^{r'})^r)$ for $r \leftarrow \mathbb{Z}_p^*$, $r' \leftarrow \mathbb{Z}_p$. He sends σ'_0, σ'_1 to the issuer.
- The user proves

$$\mathsf{ZKAK}[(usk, dsid, dsrnd, v, r'); \mathsf{Vrfy}_{\mathrm{PS}}(pp, pk_{\mathrm{PS}}, (usk, dsid, dsrnd, v), (\sigma'_0, \sigma'_1 \cdot (\sigma'_0)^{-r'})) = 1]$$

- If the proof accepts, the issuer sends $(\sigma''_0, \sigma''_1) = ((\sigma'_0)^{r''}, ((\sigma'_1) \cdot (\sigma'_0)^{y_4 \cdot k})^{r''})$ for a random $r'' \leftarrow \mathbb{Z}_p^*$ to the user.
- The user unblinds the signature as $\sigma = (\sigma_0^*, \sigma_1^*) = (\sigma''_0, \sigma''_1 \cdot (\sigma''_0)^{-r'})$ and checks $\mathsf{Vrfy}_{\mathrm{PS}}(pp, pk_{\mathrm{PS}}, (usk, dsid, dsrnd, v + k), \sigma_{\mathrm{PS}}^*) \stackrel{!}{=} 1$. If the check succeeds, it outputs $token^* = (dsid, dsrnd, v + k, \sigma_{\mathrm{PS}}^*)$, otherwise it outputs $\perp$.

$(token^*, Dsid^*) \leftarrow \mathsf{Spend}(ispp, pk, k, dsid, usk, token) \leftrightarrow \mathsf{Deduct}(ispp, pk, k, dsid, sk) \rightarrow (b, dstag)$ where $token = (dsid, dsrnd, v, cred)$ works as follows:

- The user chooses random $dsid^*_{\mathrm{usr}} \leftarrow \mathbb{Z}_p$ and computes the commitment ${C_{\mathrm{usr}}}^* = (g^{dsid^*_{\mathrm{usr}}} \cdot h^{open^*}, g^{open^*})$ for a random $open^* \leftarrow \mathbb{Z}_p$. It sends ${C_{\mathrm{usr}}}^*$ to the issuer.
- The issuer replies with a random $dsid^*_{\mathrm{isr}} \leftarrow \mathbb{Z}_p$ and a random challenge $\gamma \leftarrow \mathbb{Z}_p$. Both issuer and user compute ${C_{\mathrm{dsid}}}^* = (g^{dsid^*_{\mathrm{usr}}} \cdot h^{open^*} \cdot g^{dsid^*_{\mathrm{isr}}}, g^{open^*})$.
- The user prepares new values $dsid^* = dsid^*_{\mathrm{usr}} + dsid^*_{\mathrm{isr}}$ and $dsrnd^* \leftarrow \mathbb{Z}_p$ for his next token and computes $Dsid^* = w^{dsid^*}$ and $C = (g^{y_1})^{usk} \cdot (g^{y_2})^{dsid^*} \cdot (g^{y_3})^{dsrnd^*} \cdot (g^{y_4})^{v-k} \cdot g^{r_C}$ for a random $r_C \leftarrow \mathbb{Z}_p$.
- The user computes $c = usk \cdot \gamma + dsrnd$.
- The user encrypts $Dsid^*$ as $ctrace = (w^r, (w^r)^{usk} \cdot Dsid^*)$ for a random $r \leftarrow \mathbb{Z}_p$.
- The user randomizes his credential $(\sigma'_0, \sigma'_1) = (\sigma_0^{r''}, (\sigma_1 \cdot \sigma_0^{r'})^{r''})$ for $r'' \leftarrow \mathbb{Z}_p^*$, $r' \leftarrow \mathbb{Z}_p$
- The user sends $C, c, ctrace, \sigma'_0, \sigma'_1$ to the issuer and proves

$$\begin{aligned}
&\mathsf{ZKAK}[(usk, dsrnd, v, dsid^*, dsrnd^*, r', r, r_C, open^*);\\
&c = usk \cdot \gamma + dsrnd\\
\wedge &\mathsf{Vrfy}_{\mathrm{PS}}(pp, pk_{\mathrm{PS}}, (usk, dsid, dsrnd, v), (\sigma'_0, \sigma'_1 \cdot (\sigma'_0)^{-r'})) = 1\\
\wedge &v \geq k\\
\wedge &ctrace = (w^r, (w^r)^{usk} \cdot w^{dsid^*})\\
\wedge &C = (g^{y_1})^{usk} \cdot (g^{y_2})^{dsid^*} \cdot (g^{y_3})^{dsrnd^*} \cdot (g^{y_4})^{v-k} \cdot g^{r_C}\\
\wedge &{C_{\mathrm{dsid}}}^* = (g^{dsid^*} \cdot h^{open^*}, g^{open^*})]
\end{aligned}$$

  If the proof fails, the issuer aborts and outputs $(0, \perp)$.
- If the proof accepts, the issuer sends $\sigma''_{\mathrm{PS}} = (\sigma''_0, \sigma''_1) = (g^{r''''}, (C \cdot g^x)^{r''''})$ for a random $r'''' \leftarrow \mathbb{Z}_p^*$ to the user and outputs $(1, dstag = (c, \gamma, ctrace))$.
- The user unblinds the signature as $\sigma^*_{\mathrm{PS}} = (\sigma''_0, \sigma''_1 \cdot (\sigma''_0)^{-r_C})$.
- The user checks $\mathsf{Vrfy}_{\mathrm{PS}}(pp, pk_{\mathrm{PS}}, (usk, dsid^*, dsrnd^*, v - k), \sigma^*_{\mathrm{PS}}) \stackrel{!}{=} 1$. If the check succeeds, it outputs $token^* = (dsid^*, dsrnd^*, v - k, \sigma^*_{\mathrm{PS}})$ and $Dsid^*$, otherwise it outputs $\perp$.

$\mathsf{Link}(ispp, dstag, dstag') \rightarrow (upk, dslink)$ given $dstag = (c, \gamma, ctrace)$ and $dstag' = (c', \gamma', ctrace')$, outputs $dslink = (c - c')/(\gamma - \gamma')$ and $upk = w^{dslink}$.

$\mathsf{Trace}(ispp, dslink, dstag) \rightarrow Dsid^*$ for $dstag = (c, \gamma, (ctrace_0, ctrace_1))$ computes $Dsid^* = ctrace_1 \cdot ctrace_0^{-dslink}$.

$\mathsf{VrfyDs}(ispp, dslink, upk) \rightarrow b$ outputs 1 iff $w^{dslink} = upk$.

D Security Proofs for the Incentive System

D.1 Correctness in the Presence of Adversarial Users

For completeness, we define correctness in the presence of adversarial users here, which rules out that adversarial users can interfere with operations between honest users and an honest issuer.

Definition 24 (Correctness in the presence of adversarial users)**.** Let Π be an incentive system. Consider the following experiment $\text{Exp}^{\text{adv-corr}}(\Pi, \mathcal{A}, \lambda)$:

- The experiment sets up $ispp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda))$ and calls the oracle $pk \leftarrow \textbf{IssuerKeyGen}()$. It hands $ispp$ and pk to $\mathcal{A}$.
- $\mathcal{A}$ may query the following oracles
  - $upk \mapsto \textbf{Issue}(upk)$
  - $k \mapsto \textbf{Credit}(k) \leftrightarrow \mathcal{A}$
  - $k \mapsto \textbf{Deduct}(k) \leftrightarrow \mathcal{A}$
  - $s \mapsto \textbf{DBsync}(s)$
  - $\textbf{Keygen}()$
  - $u \mapsto \textbf{Join}(u, pk) \leftrightarrow \textbf{Issue}(upk_u)$
  - $(u, k) \mapsto \textbf{Earn}(u, k) \leftrightarrow \textbf{Credit}(k)$
  - $(u, k) \mapsto \textbf{Spend}(u, k) \leftrightarrow \textbf{Deduct}(k)$
- Eventually, $\mathcal{A}$ halts.
- The experiment outputs 1 iff $\mathcal{DB}$ contains some $dsid_u$ of some honest user u (i.e. user u's next spend operation would be detected double-spending as $dsid_u$ is already in $\mathcal{DB}$).

We say that Π has *correctness in the presence of adversarial users* if for all ppt $\mathcal{A}$, there exists a negligible function *negl* s.t. $\Pr[\text{Exp}^{\text{adv-corr}}(\Pi, \mathcal{A}, \lambda) = 1] \leq negl(\lambda)$ for all λ.

Note that correctness in the presence of adversarial users is not implied by correctness, soundness and framing resistance. Correctness does not imply anything for the case in which there are adversarial users. Framing resistance implies that u cannot be blamed for the double-spending (it may still happen that the online double-spending prevention prevents u from spending his coins). Soundness implies that after u spends his coin, *someone* can be blamed for it. This does not rule out that a corrupted user is able to inject u's *dsid* into $\mathcal{DB}$ while taking the blame. However, this would essentially constitute a denial of service attack on u, which is why correctness in the presence of adversarial users is a desirable property.

Theorem 25. If $\mathbb{Z}_p$ is super-poly, then Π_{InSy} (Construction 14) is correct in the presence of adversarial users (Definition 24).

Proof. Assume there are k *dsid* entries in $\mathcal{DB}$ and ℓ honest users u at the point where $\mathcal{A}$ halts. For honest users, $dsid_u$ is uniformly random in $\mathbb{Z}_p$ by construction. Furthermore, $\mathcal{A}$'s view is independent of the current $(dsid_u)_{u \text{ honest}}$ as none of the oracles output any information about them. So the probability that some $dsid_u$ is one of the k *dsid* in $\mathcal{DB}$ is at most $\ell \cdot k/|\mathbb{Z}_p|$, which is negligible as ℓ and k are polynomial and $|\mathbb{Z}_p|$ is super-poly. □

D.2 Incentive System Anonymity

In the following we proof Theorem 15. For the proof of the theorem we have to look at the experiment $\text{Exp}^{\text{anon-X}}$ (Fig. 3) instantiated for the incentive system Π_{InSy}. On a high level, in Π_{InSy} the important information for anonymity are the user specific values. Ignoring the commitments and ciphertexts, we could solely rely on the simulatability of the protocols to proof the theorem. However, the commitment and encryption scheme only guarantees computationally hiding and key-ind. CPA security.

Lemma 26. If $\Pi_{\mathcal{C}}$ has simulation anonymity (Definition 5), then for all ppt algorithms $\mathcal{A}$ it holds that $|\Pr[\mathrm{Exp}_0^{\text{anon-}\mathsf{Earn}}(\Pi_{\mathrm{InSy}}, \mathcal{A}, \lambda) = 1] - \Pr[\mathrm{Exp}_1^{\text{anon-}\mathsf{Earn}}(\Pi_{\mathrm{InSy}}, \mathcal{A}, \lambda) = 1]| = 0$.

Proof. We have to show that we can simulate the experiment and especially the challenge phase independent of b. Since $\Pi_{\mathcal{C}}$ satisfies simulation anonymity (Definition 5), there are ppt algorithms $\mathfrak{S}_{\mathsf{Setup}}, \mathfrak{S}_{\mathsf{Receive}}, \mathfrak{S}_{\mathsf{ShowPrv}}, \mathfrak{S}_{\mathsf{UpdRcv}}$. Therefore, we can perfectly simulate the setup by running $\mathfrak{S}_{\mathsf{Setup}}$. Next, observe that we can honestly execute the oracles as in the experiment, since we know all inputs of the users. In the challenge phase the experiment executes $\mathbf{Earn} \leftrightarrow \mathcal{A}$, where $\mathbf{Earn}$ is an execution of $\mathsf{UpdRcv}_{\mathcal{C}} \leftrightarrow \mathcal{A}$. We can perfectly simulate $\mathbf{Earn}$ in the challenge phase independent of b by running $\mathfrak{S}_{\mathsf{UpdRcv}} \leftrightarrow \mathcal{A}$. □

In the Spend case of experiment $\mathrm{Exp}_b^{\text{anon-}\mathsf{Spend}}(\Pi_{\mathrm{InSy}}, \mathcal{A}, \lambda)$ we have to look at the $\mathsf{Spend} \leftrightarrow \mathsf{Deduct}$ protocol of Π_{InSy} (Construction 14), since the setup and challenge phase of $\mathrm{Exp}_b^{\text{anon-}\mathsf{Spend}}$ executes $\mathsf{Spend}(u_b, k) \leftrightarrow \mathcal{A}$. In the challenge phase the adversary $\mathcal{A}$ is asked to guess which of the users u_0, u_1 that he picked before executed the Spend protocol.

Let us first state where $\Pi_{\mathcal{C}^+}$ and $\Pi_{\mathcal{E}}$ are used. During $\mathsf{Spend} \leftrightarrow \mathsf{Deduct}$ the issuer (in $\mathrm{Exp}_b^{\text{anon-}\mathsf{Spend}}$ the adversary) gets commitments (generated with $\Pi_{\mathcal{C}^+}$) from the users during the combined generation of a fresh $dsid^* = dsid^*_{\mathrm{usr}} + dsid^*_{\mathrm{isr}}$, where the user commits to a $dsid^*_{\mathrm{usr}} \leftarrow \mathbb{Z}_p$ and the issuer provides his $dsid^*_{\mathrm{isr}} \in \mathbb{Z}_p$.

Also during $\mathsf{Spend} \leftrightarrow \mathsf{Deduct}$, the user encrypts $dsid^*$ under his user public key $upk_{\mathcal{E}}$ as $ctrace \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(pp, upk_{\mathcal{E}}, dsid^*)$. Here the adversary could break anonymity by distinguishing which user public key was used to encrypt or the breaks CPA security.

Remember that the adversary $\mathcal{A}$ in $\mathrm{Exp}_b^{\text{anon-}\mathsf{Spend}}(\Pi_{\mathrm{InSy}}, \mathcal{A}, \lambda)$ can query the spend oracle $\mathbf{Spend}(u, k)$ for user u and spend value k in the setup and challenge phase. In each of the oracle executions he learns the $dsid$ of the token that the user spends. This means that Spend executions in the setup phase and the execution in the challenge phase are implicitly linked. In detail, $\mathcal{A}$ chooses users u_0, u_1 in the challenge phase. Then in the challenge phase Spend, $\mathcal{A}$ learns the $dsid^*_b$ to a commitment C^*_b and encryption $ctrace_b$ he received during the last Spend execution in the setup phase with either u_0 or u_1. If he could link the information, he would break anonymity. Let us quickly deal with the easy case where $\mathcal{A}$ never triggered a spend operation during the setup phase, then the $dsid^*$ that he gets during the challenge Spend is a fresh random value from $\mathbb{Z}_p$ w.h.p..

For the rest of the proof we will change the challenge phase. In detail, we change in the challenge Spend execution which $dsid^*_b$ the adversary $\mathcal{A}$ gets (index i in the following) and how the encryption $ctrace$ that $\mathcal{A}$ receives is generated (index j in the following). Therefore, we define experiments $H_{i,j}$ where $i, j \in \{0, 1\}$.

Let $H_{0,0} = \mathrm{Exp}_0^{\text{anon-}\mathsf{Spend}}(\Pi_{\mathrm{InSy}}, \mathcal{A}, \lambda)$ and $H_{1,1} = \mathrm{Exp}_1^{\text{anon-}\mathsf{Spend}}(\Pi_{\mathrm{InSy}}, \mathcal{A}, \lambda)$. In $H_{0,0}$ the adversary gets in the challenge phase one execution of Spend with user u_0 where $\mathcal{A}$ receives $dsid^*_0$ ($j = 0$). Therefore, the only important Spend execution of the setup phase is the last execution with user u_0 ($i = 0$) where $\mathcal{A}$ gets the commitment $C^*_0 = \mathsf{Commit}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, dsid^*_{\mathrm{usr}})$ and encryption $ctrace_0 = \mathsf{Encrypt}_{\mathcal{E}}(pp, upk_{\mathcal{E},0}, dsid^*_0)$, where $dsid^*_0 = dsid^*_{\mathrm{usr}} + dsid^*_{\mathrm{isr}}$. $H_{1,1}$ is analogous. To show $|[\Pr[H_{0,0} = 1] - \Pr[H_{1,1} = 1]]| \leq negl$ we also define an intermediate experiment $H_{0,1}$ and prove that $|\Pr[H_{0,0} = 1] - \Pr[H_{0,1} = 1]| \leq negl$ and $|\Pr[H_{0,1} = 1] - \Pr[H_{1,1} = 1]| \leq negl$. In $H_{0,1}$ we output $dsid^*_0$ in the challenge Spend execution, where $dsid^*_0$ was determined and used in the previous Spend execution (part of the setup phase) with user u_0. The change is that we no longer also output an encryption of $dsid^*_0$ under $upk_{\mathcal{E},0}$. Instead, we output $ctrace' = \mathsf{Encrypt}_{\mathcal{E}}(pp, upk_{\mathcal{E},1}, dsid^*_1)$, where $dsid^*_1$ was determined and used in the previous Spend execution (part of the setup phase) with user u_1. The public key $upk_{\mathcal{E},1}$ is also the one of user u_1.

Lemma 27. If $\Pi_{\mathcal{C}}$ has simulation anonymity and $\Pi_{\mathcal{E}}$ is key-ind. CPA secure, then for all ppt $\mathcal{A}$, $|(\Pr[H_{0,0}(\mathcal{A}) = 1] - \Pr[H_{0,1}(\mathcal{A}) = 1])| = negl(\lambda)$ for all $\lambda \in \mathbb{N}$.

Proof. Assume that adversary $\mathcal{A}$ distinguishes $H_{0,0}, H_{0,1}$ with non-negligible probability. We give a reduction B using $\mathcal{A}$ to key-indistinguishable CPA security (Definition 20) of $\Pi_{\mathcal{E}}$. In the reduction B we get from $\text{Exp}_b^{\text{key-ind}}(\Pi_{\mathcal{E}}, \mathcal{A}, \lambda)$ ($b \in \{0,1\}$) two public keys that B injects as two user public keys $upk_{\mathcal{E},0}$ and $upk_{\mathcal{E},1}$ by guessing one pair of the users that $\mathcal{A}$ can choose in the challenge phase of $H_{0,b}$. Since $\Pi_{\mathcal{C}}$ guarantees simulation anonymity (Definition 5) and $c = usk \cdot \gamma + dsrnd$ is perfectly hiding, the reduction B can simulate the setup of the incentive system and the oracles **Keygen**, **Join**, **Earn**, **Spend** of $H_{0,b}$ for two users u_0, u_1 that we choose before. For all other users B executes the oracles honestly as in the experiment. If $\mathcal{A}$ outputs two users that are not our guess u_0, u_1, then B aborts. This happens with probability $1 - \frac{1}{poly(\lambda)^2}$. Otherwise, in the challenge phase with $\mathcal{A}$, Spend is changed in B as described above. In detail, B gives the $\text{Exp}_b^{\text{key-ind}}$ challenger $dsid_0^*$ and $dsid_1^*$ (both from the latest token of the users u_0, u_1 from the setup phase) and outputs the answer of the challenger as the encryption for $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs his guess $\hat{b}$ which B outputs to the $\text{Exp}_b^{\text{key-ind}}$ challenger. Consequently, $|\Pr[\text{Exp}_0^{\text{key-ind}}(\Pi_{\mathcal{E}}, B, \lambda) = 1] - \Pr[\text{Exp}_1^{\text{key-ind}}(\Pi_{\mathcal{E}}, B, \lambda) = 1]| = \frac{1}{poly(\lambda)^2} \cdot |(\Pr[H_{0,0}(\mathcal{A}) = 1] - \Pr[H_{0,1}(\mathcal{A}) = 1])|$. □

Next, we look at $|\Pr[H_{0,1} = 1] - \Pr[H_{1,1} = 1]|$. From $H_{0,1}$ to $H_{1,1}$ we change which $dsid_b^*$ the adversary receives during the challenge Spend execution. Either $dsid_0^*$ that is part of the latest token of the user u_0 or $dsid_1^*$ from the latest token of user u_1. As described above the adversary receives commitments and encryptions for $dsid_b^*$ corresponding to the latest token of the users. Remember, $H_{1,1} = \text{Exp}_1^{\text{anon-}\mathsf{Spend}}(\Pi_{\text{InSy}}, \mathcal{A}, \lambda)$.

Lemma 28. If $\Pi_{\mathcal{C}}$ guarantees simulation anonymity, $\Pi_{\mathcal{E}}$ is key-indistinguishable CPA secure, $\Pi_{\mathcal{C}^+}$ is computational hiding, then for all ppt adversaries $\mathcal{A}$ it holds that $|(\Pr[H_{0,1}(\mathcal{A}) = 1] - \Pr[H_{1,1}(\mathcal{A}) = 1])| = negl$

Lemma 28 follows from the following lemmas. First, we define a helper experiment $G_{u,v,x,y}^b(D, \lambda)$ for $u, v, x, y \in \{0,1\}$ that we will use in the following lemmas.

$\mathbf{G}_{\mathbf{u},\mathbf{v},\mathbf{x},\mathbf{y}}^{\mathbf{b}}(\mathbf{D}, \lambda)$:

- $pp \leftarrow \mathcal{G}(1^\lambda)$
- $pk_{\mathcal{C}^+} \leftarrow \mathsf{KeyGen}_{\mathcal{C}^+}(pp)$
- $sk_0, sk_1 \leftarrow \mathsf{KeyGen}_{\mathcal{E}}()$ and $pk_0, pk_1 \leftarrow \mathsf{ComputePK}_{\mathcal{E}}(pp)$
- Hand D $pp, pk_{\mathcal{C}^+}, pk_0$, and pk_1
- Choose two messages $m_0, m_1 \leftarrow M_{pp}$

Phase 1:

- Hand D the commitment C_u where $\mathsf{Commit}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, m_u) \rightarrow (C_u, \mathsf{Open})$
- Receive $\mathsf{share} \in M_{pp}$ from D
- Hand D the encryption $S_v \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(pp, pk_v, m_v + \mathsf{share})$

Phase 2:

- Hand D the commitment C_x where $\mathsf{Commit}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, m_x) \rightarrow (C_x, \mathsf{Open})$
- Receive $\mathsf{share}' \in M_{pp}$ from D
- Hand D the encryption $S_y \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(pp, pk_y, m_y + \mathsf{share}')$

Challenge:

- Hand D message m_b
- Receive $\hat{b}$ from D
- Output 1 iff $\hat{b} = b$

Lemma 29. If $\Pi_{\mathcal{C}}$ guarantees simulation anonymity, $\Pi_{\mathcal{E}}$ is key-indistinguishable CPA secure, $\Pi_{\mathcal{C}^+}$ is computational hiding, then there is an ppt reduction D such that for all ppt adversaries $\mathcal{A}$ it holds that $|\Pr[G_{0,0,1,1}^0(D, \lambda) = 1] - \Pr[G_{0,0,1,1}^1(D, \lambda) = 1]| = \frac{1}{poly\lambda} \cdot |\Pr[H_{0,1}(\mathcal{A}) = 1] - \Pr[H_{1,1}(\mathcal{A}) = 1]|$.

Proof. Assume that an adversary $\mathcal{A}$ distinguishes $H_{0,1}$ and $H_{1,1}$, then we can give an reduction D that distinguishes $G^0_{0,0,1,1}(D,\lambda)$ and $G^1_{0,0,1,1}(D,\lambda)$.

In the following we define D against $G^b_{0,0,1,1}(D,\lambda)$ using $\mathcal{A}$. To shorten the proof, in D the guessing of two users u_0, u_1 to inject the public keys given by $G^b_{0,0,1,1}(D,\lambda)$ and the handling of the oracle queries is analogous to B. The last **Spend** query by $\mathcal{A}$ for user u_0 is answered with the help of Phase 1 in $G^b_{0,0,1,1}(D,\lambda)$ and the rest of Spend simulated. From Phase 1 D uses the commitment C_u instead of generating a commitment to a fresh $dsid_{\text{usr}}$. In Spend, $\mathcal{A}$ hands D (acting as the user) a $dsid_{\text{isr}}$ that D hands itself to $G^b_{0,0,1,1}(D,\lambda)$ (Phase 1) as share. The encryption that D gets from Phase 1 is used as the encryption *ctrace* in Spend. For the last **Spend** query by $\mathcal{A}$ for user u_1 reduction D acts analogous with the difference that D uses Phase 2. Eventually $\mathcal{A}$ enters the challenge phase and outputs two user handles. If the handles are not the one that D guessed, then abort. Otherwise, D simulates a Spend with $\mathcal{A}$ where D is supposed to send $\mathcal{A}$ the *dsid* of the latest token of the challenged user. Hence, D sends the message m_b that D received in the challenge phase of $G^b_{0,0,1,1}(D,\lambda)$ instead. If $\mathcal{A}$ outputs $\hat{b}$, D also outputs $\hat{b}$ to $G^b_{0,0,1,1}(D,\lambda)$. Overall, $|\Pr[G^0_{0,0,1,1}(D,\lambda)=1]-\Pr[G^1_{0,0,1,1}(D,\lambda)=1]| = \frac{1}{poly\lambda}\cdot|\Pr[H_{0,1}(\mathcal{A})=1]-\Pr[H_{1,1}(\mathcal{A})=1]|$. □

It is left to show that for all ppt algorithms E it holds that $|\Pr[G^0_{0,0,1,1}(E,\lambda)=1]-\Pr[G^1_{0,0,1,1}(E,\lambda)=1]| \leq negl$. Remember, in $G^b_{u,v,x,y}(D,\lambda)$ the bits (u,v) determine Phase 1, (x,y) Phase 2, and b determines the output message m_b at the end of the experiment. In detail, the bits u and x determine the messages for the commitments; the bits v and y determine the messages and public keys for the encryption. In Figure 5 we show an overview of the following proof steps, where Phase 1 and Phase 2 points to the point where we introduce a change to the previous game and "key-ind. CPA" respectively "comp. hiding" is the security guarantee that we use in the reduction.

Lemma 30. It holds that $G^0_{1,1,0,0} = G^1_{0,0,1,1}$.

Proof. This is the last step presented in Figure 5. The lemma follows from the following observation. Since the experiment $G_{u,v,x,y}$ chooses the challenge messages itself, the order of the Phases can be switched without changing the game while also changing the challenge message from m_0 to m_1. Changing order of the Phases is the same as replacing m_u, m_v, m_x, and m_y by m_{1-u}, m_{1-v}, m_{1-x}, and m_{1-y}. □

Lemma 31. If $\Pi_{\mathcal{E}}$ is key-indistinguishable CPA secure, then for all ppt adversaries E it holds that
$|\Pr[G^0_{0,0,1,1}(E,\lambda)=1]-\Pr[G^0_{0,1,1,1}(E,\lambda)=1]| = negl(\lambda)$.

Proof. We show that if there is an adversary E s.t. $|\Pr[G^0_{0,0,1,1}(E,\lambda)=1]-\Pr[G^0_{0,1,1,1}(E,\lambda)=1]|$ is non-negligible, than we can give an reduction $R^{\text{Phase 1}}_{\text{ki-cpa}}$ that breaks key-ind. CPA (Definition 20, $\text{Exp}^{\text{key-ind}}_b(\Pi_{\mathcal{E}}, R^{\text{Phase 1}}_{\text{ki-cpa}}, \lambda)$) using E.

$R^{\text{Phase 1}}_{\text{ki-cpa}}$ gets from its experiment $\text{Exp}^{\text{key-ind}}_b$ public parameters pp and two encryption scheme public keys pk_0, pk_1. $R^{\text{Phase 1}}_{\text{ki-cpa}}$ generates honestly a commitment public key $pk_{\mathcal{C}^+}$ and hands E $pk_{\mathcal{C}^+}$ and the received pp, pk_0, pk_1 as in $G^0_{0,v,1,1}$. Next, $R^{\text{Phase 1}}_{\text{ki-cpa}}$ chooses two messages $m_0, m_1 \leftarrow M_{pp}$, generates a commitment to m_0 for Phase 1 as in $G^0_{0,v,1,1}$. $R^{\text{Phase 1}}_{\text{ki-cpa}}$ sends the Phase 1 commitment to E and gets share $\in M_{pp}$ back. Next, $R^{\text{Phase 1}}_{\text{ki-cpa}}$ hands $m^*_0 = m_0 + \text{share}$ and $m^*_1 = m_1 + \text{share}$

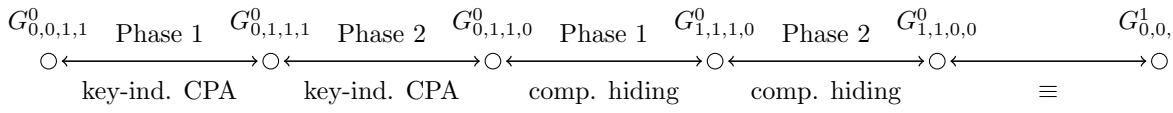

Figure 5: Sequence of games for anonymity proof

to $\text{Exp}_b^{\text{key-ind}}$ and gets $S \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(pp, pk_b, m_b^*)$ back. Phase 2 is executed by $R_{\text{ki-cpa}}^{\text{Phase 1}}$ as in the experiment $G_{0,v,1,1}^0$ (commitment to m_1 and encryption of $m_1 + \mathsf{share}'$). In the challenge phase, $R_{\text{ki-cpa}}^{\text{Phase 1}}$ hands the message m_0 to E and receives E's guess $\hat{b}$ that $R_{\text{ki-cpa}}^{\text{Phase 1}}$ also outputs to $\text{Exp}_b^{\text{key-ind}}$.

Observe that $R_{\text{ki-cpa}}^{\text{Phase 1}}$ perfectly simulates the view of E in $G_{0,0,1,1}^0$ if $R_{\text{ki-cpa}}^{\text{Phase 1}}$ acts in the experiment $\text{Exp}_0^{\text{key-ind}}$. The same holds for the view of E in $G_{0,1,1,1}^0$ if $R_{\text{ki-cpa}}^{\text{Phase 1}}$ acts in the experiment $\text{Exp}_1^{\text{key-ind}}$. Consequently, $|\Pr[\text{Exp}_0^{\text{key-ind}}(\Pi_{\mathcal{E}}, R_{\text{ki-cpa}}^{\text{Phase 1}}, \lambda) = 1] - \Pr[\text{Exp}_1^{\text{key-ind}}(\Pi_{\mathcal{E}}, R_{\text{ki-cpa}}^{\text{Phase 1}}, \lambda) = 1]| = |\Pr[G_{0,0,1,1}^0(\mathcal{E}, \lambda) = 1] - \Pr[G_{0,1,1,1}^0(\mathcal{E}, \lambda) = 1]|$. □

Lemma 32. If $\Pi_{\mathcal{E}}$ is key-indistinguishable CPA secure, then for all ppt adversaries E it holds that
$|\Pr[G_{0,1,1,1}^0(E, \lambda) = 1] - \Pr[G_{0,1,1,0}^0(E, \lambda) = 1]| = negl(\lambda)$.

The reduction $R_{\text{ki-cpa}}^{\text{Phase 2}}$ to show the above lemma works analogous to $R_{\text{ki-cpa}}^{\text{Phase 1}}$ with the difference that in $R_{\text{ki-cpa}}^{\text{Phase 2}}$ we use the encryption challenge of $\text{Exp}_b^{\text{key-ind}}(\Pi_{\mathcal{E}}, R_{\text{ki-cpa}}^{\text{Phase 1}}, \lambda)$ in Phase 2.

Lemma 33. If $\Pi_{\mathcal{C}^+}$ is computational hiding, then for all ppt adversaries F it holds that $|\Pr[G_{0,1,1,0}^0(F, \lambda) = 1] - \Pr[G_{1,1,1,0}^0(F, \lambda) = 1]| = negl$.

Proof. In the following we show that if there is an adversary F such that $|\Pr[G_{0,1,1,0}^0(F, \lambda) = 1] - \Pr[G_{1,1,1,0}^0(F, \lambda) = 1]|$ is non-negligible, than we can give an reduction $R_{\text{hiding}}^{\text{Phase 1}}$ that breaks comp. hiding of the commitment scheme $\Pi_{\mathcal{C}^+}$ (Definition 22, $\text{Exp}_b^{\text{hiding}}(\Pi_{\mathcal{C}^+}, R_{\text{hiding}}^{\text{Phase 1}}, \lambda)$) using F.

$R_{\text{hiding}}^{\text{Phase 1}}$ gets from its experiment $\text{Exp}_b^{\text{hiding}}$ public parameters pp and a commitment scheme public key $pk_{\mathcal{C}^+}$. $R_{\text{hiding}}^{\text{Phase 1}}$ generates honestly two encryption public keys pk_0, pk_1 as in $G_{u,1,1,0}^0$. hands F pk_0, pk_1 and the received pp, pk as in $G_{u,1,1,0}^0$. Next, $R_{\text{hiding}}^{\text{Phase 1}}$ chooses two messages $m_0, m_1 \leftarrow M_{pp}$, hands both to $\text{Exp}_b^{\text{hiding}}$, and gets $C_b \leftarrow \mathsf{Commit}_{\mathcal{C}^+}(pp, pk_{\mathcal{C}^+}, m_b)$ back. $R_{\text{hiding}}^{\text{Phase 1}}$ outputs C_b as the Phase 1 commitment to F. The rest of Phase 1 and Phase 2 are executes by $R_{\text{hiding}}^{\text{Phase 1}}$ honestly as in the experiment $G_{u,1,1,0}^0$. In the challenge phase, $R_{\text{hiding}}^{\text{Phase 1}}$ hands F the message m_b and receives F's guess $\hat{b}$. $R_{\text{hiding}}^{\text{Phase 1}}$ also outputs $\hat{b}$ to $\text{Exp}_b^{\text{hiding}}$. Observe that $R_{\text{hiding}}^{\text{Phase 1}}$ perfectly simulates the view of F in $\text{Exp}_b^{\text{hiding}}$. Consequently, $|\Pr[\text{Exp}_0^{\text{hiding}}(\Pi_{\mathcal{C}^+}, R_{\text{hiding}}^{\text{Phase 1}}, \lambda) = 1] - \Pr[\text{Exp}_1^{\text{hiding}}(\Pi_{\mathcal{C}^+}, R_{\text{hiding}}^{\text{Phase 1}}, \lambda) = 1]| = |\Pr[G_{0,1,1,0}^0(F, \lambda) = 1] - \Pr[G_{1,1,1,0}^0(F, \lambda) = 1]|$. □

Lemma 34. If $\Pi_{\mathcal{C}^+}$ is computational hiding, then for all ppt adversaries F it holds that $|\Pr[G_{1,1,1,0}^0(F, \lambda) = 1] - \Pr[G_{1,1,0,0}^0(F, \lambda) = 1]| = negl$.

The reduction $R_{\text{hiding}}^{\text{Phase 2}}$ to show the above lemma works analogous to $R_{\text{hiding}}^{\text{Phase 1}}$ with the difference that in $R_{\text{hiding}}^{\text{Phase 2}}$ we use the commitment challenge of $\text{Exp}_b^{\text{hiding}}(\Pi_{\mathcal{E}}, R_{\text{hiding}}^{\text{Phase 2}}, \lambda)$ in Phase 2.

This concludes the proof of Lemma 28 and therefore of Theorem 15.

D.3 Incentive System Soundness

Theorem 16. Let $\mathcal{A}$ be an attacker against incentive system soundness of Construction 14. We construct $\mathcal{B}$ against updatable credential soundness of $\Pi_{\mathcal{C}}$.

- $\mathcal{B}$ receives cpp from its challenger. $\mathcal{B}$ replies with 1^4 to receive pk. It completes the setup by choosing $pk_{\mathcal{C}^+} \leftarrow \mathsf{KeyGen}_{\mathcal{C}^+}(pp)$. Then $\mathcal{B}$ simulates the query to **IssuerKeyGen**(): instead of running $\mathsf{IssuerKeyGen}$, $\mathcal{B}$ uses its challenger's key pk as the query result. $\mathcal{B}$ outputs $ispp = (pp, cpp, pk_{\mathcal{C}^+})$ and pk to $\mathcal{A}$.
- Oracle queries by $\mathcal{A}$ are simulated by $\mathcal{B}$ as prescribed by the protocol with one exception: whenever the original protocol would run $\mathsf{Issue}_{\mathcal{C}}$ or $\mathsf{Update}_{\mathcal{C}}$, $\mathcal{B}$ instead queries its challenger for the corresponding operation and relays protocol messages between the challenger and $\mathcal{A}$.

- Eventually $\mathcal{A}$ halts. Then $\mathcal{B}$ halts as well.

Obviously, the view of $\mathcal{A}$ is the same whether it interacts with the incentive system soundness challenger or with $\mathcal{B}$.

Let error be the event that (1) $\mathcal{B}$ has output the same challenge δ in two different **Deduct** runs, or (2) there were two commitments $C_{\text{dsid}}, C_{\text{dsid}}{}'$ in two runs of **Deduct** or **Issue** such that the commitments can be opened to two different messages. Note that $\mathcal{C}^+$ is perfectly binding by assumption and so every commitment opens to at most one value (which $\mathcal{B}$ cannot necessarily efficiently compute, but as an event, this is well-defined).

It holds that $\Pr[\text{error}] \leq \mathit{negl}(\lambda)$ because (1) in each **Deduct** query, δ is chosen uniformly random by $\mathcal{B}$ from the super-poly size set $\mathbb{Z}_p$, and (2) C_{dsid} is the result of an $\mathsf{Add}_{\mathcal{C}^+}$ operation with a uniformly random value $\mathit{dsid}_{\text{isr}} \leftarrow \mathbb{Z}_p$ chosen by $\mathcal{B}$, hence it opens (only) to a uniformly random value.

Let $\mathcal{A}\text{wins}_{\text{trace}}$ be the event that $\mathcal{DB}$ contains some (upk, dslink) s.t. $\mathsf{VrfyDs}(\mathit{ispp}, \mathit{dslink}, \mathit{upk}) \neq 1$ or $\mathit{upk} \notin \mathcal{U}$. Let $\mathcal{A}\text{wins}_{\text{overspend}}$ be the event that $v_{\text{spent}} - v_{\text{invalid}} > v_{\text{earned}}$ and $\mathsf{DBsync}(s)$ has been queried for all spend handles s. Let $\mathcal{A}\text{wins}$ be the event that $\mathcal{A}$ wins the game, $\mathcal{A}\text{wins}_{\text{trace}} \vee \mathcal{A}\text{wins}_{\text{overspend}}$. Lemma 35 will show that if $\mathcal{A}\text{wins} \wedge \neg\text{error}$ occurs, then there *exists* no explanation list $\mathcal{L}$ that is consistent, implying $\Pr[\text{Exp}^{\text{sound}}(\Pi, \mathcal{B}, \mathcal{E}, \lambda) = 1 \mid \mathcal{A}\text{wins} \wedge \neg\text{error}] = 1$. Overall, let $\mathcal{E}$ be an algorithm, then

$$
\begin{aligned}
&\Pr[\text{Exp}^{\text{sound}}(\Pi_{\mathcal{C}}, \mathcal{B}, \mathcal{E}, \lambda) = 1] \\
\geq &\Pr[\text{Exp}^{\text{sound}}(\Pi_{\mathcal{C}}, \mathcal{B}, \mathcal{E}, \lambda) = 1 \mid \mathcal{A}\text{wins} \wedge \neg\text{error}] \cdot \Pr[\mathcal{A}\text{wins} \wedge \neg\text{error}] \\
= &1 \cdot \Pr[\mathcal{A}\text{wins} \wedge \neg\text{error}] \geq \Pr[\mathcal{A}\text{wins}] - \Pr[\text{error}] \\
= &\Pr[\text{Exp}^{\text{sound}}(\Pi_{\text{InSy}}, \mathcal{A}, \lambda) = 1] - \Pr[\text{error}].
\end{aligned}
$$

Consequently, because $\Pr[\text{Exp}^{\text{sound}}(\Pi_{\mathcal{C}}, \mathcal{B}, \mathcal{E}, \lambda) = 1]$ is negligible by assumption, it follows that $\Pr[\text{Exp}^{\text{sound}}(\Pi_{\text{InSy}}, \mathcal{A}, \lambda) = 1]$ must be negligible. □

Lemma 35. If $\mathcal{A}\text{wins} \wedge \neg\text{error}$, then no explanation list is consistent (cf. Theorem 16 and Definition 6).

Proof. We prove the statement by showing that if $\neg$error and there exists a consistent explanation list $\mathcal{L}$, then $\neg\mathcal{A}$wins. Let $\mathcal{L}$ be a consistent explanation list and let E_i be the corresponding sets of explained attribute vectors (cf. Definition 6).

For ease of reasoning in all the following lemmas, we represent the explanation list as a bipartite directed graph G (cf. Figure 6). The graph contains (1) one node $\vec{A}$ for every explained attribute vector $\vec{A} \in \bigcup_i E_i$ and (2) nodes for **Issue**, **Credit**, **Deduct** queries: If the ith query is an **Issue**(upk) query, there is a node i. If the ith query is a **Credit**(k) query for which the $\mathsf{Update}_{\mathcal{C}}$ operation outputs 1 for the issuer, there is a node i. If the ith query is an $s \leftarrow$ **Deduct**(k) query for which the $\mathsf{Update}_{\mathcal{C}}$ operation outputs 1 for the issuer, there is a node i.

An **Issue** node i has an outgoing edge to the attribute vector $\psi_i(\perp, \alpha_i)$, where ψ_i is the update function used within the ith query and α_i is as supplied by $\mathcal{L}$. A **Credit** or **Deduct** node i has an incoming edge from attribute vector $\vec{A}_i$ and an outgoing edge to $\psi_i(\vec{A}_i, \alpha_i)$, where ψ_i is the update function used within this query and $\vec{A}_i, \alpha_i$ are as supplied by $\mathcal{L}$. We call $\vec{A}_i$ the *predecessor* and $\psi_i(\vec{A}_i, \alpha_i)$ the *successor* of a **Credit Deduct** node i.

We say that a **Deduct** node i is marked *invalid* if its corresponding transaction in the double-spend database $\mathcal{DB}$ is marked invalid. Otherwise, the node is *valid*.

Lemma 36 shows that $\neg\mathcal{A}\text{wins}_{\text{trace}}$ and Lemma 37 shows that $\neg\mathcal{A}\text{wins}_{\text{overspend}}$. Hence $\neg\mathcal{A}$wins. □

For all of the following lemmas, we are in the setting of Lemma 35, i.e. we assume that $\neg$error happens and $\mathcal{L}$ is consistent.

Lemma 36. $\neg\mathcal{A}\text{wins}_{\text{trace}}$ holds (i.e. $\mathcal{DB}$ contains no $(upk, dslink)$ with $\mathsf{VrfyDs}(ispp, dslink, upk) \neq 1$ or $upk \notin \mathcal{U}$)

Proof. First, note that any output $(upk, dslink)$ of Link by definition fulfills the VrfyDs check. The "$upk \in \mathcal{U}$" part remains to be shown. Assume that at some point, $(upk, dslink)$ was associated with $dsid$ in $\mathcal{DB}$. Lemma 45 states there exists a node $\vec{A} = (usk, dsid, dsrnd, \cdot)$ in G with $usk = dslink$.

Lemma 41 implies that $\vec{A}$ is reachable from some **Issue** node. Let upk' be the public key added to $\mathcal{U}$ by the **Issue** oracle call. Let $\vec{A}' = (usk', dsid', dsrnd', 0)$ be the successor to that **Issue** node. Since $\vec{A}$ is reachable from $\vec{A}'$, we have $usk' = usk$ (no update function ever changes the user secret). Because the Issue update function checks $\mathsf{ComputePK}_{\mathcal{E}}(pp, usk') \stackrel{!}{=} upk'$, and $upk = \mathsf{ComputePK}(pp, dslink) = \mathsf{ComputePK}(pp, usk)$ by definition of DBsync we have $upk = upk'$. So it holds that upk was added to $\mathcal{U}$. □

Lemma 37. $\neg\mathcal{A}\text{wins}_{\text{overspend}}$ holds (i.e. $v_{\text{spent}} - v_{\text{invalid}} \leq v_{\text{earned}}$).

Proof. Assume that $\mathbf{DBsync}(s)$ has been queried for all spend handles s. For any subgraph H of G, we define $v_{\text{spent}}(H) = \sum_{(i,\mathbf{Deduct})\in H} k_i$, $v_{\text{earned}}(H) = \sum_{(i,\mathbf{Credit})\in H} k_i$, and $v_{\text{invalid}}(H) = \sum_{(i,\mathbf{Credit})\in H; i \text{ invalid}} k_i$. Note that these are consistent with $v_{\text{spent}}, v_{\text{earned}}, v_{\text{invalid}}$ in the incentive system soundness game, i.e. $v_{\text{spent}} = v_{\text{spent}}(G), v_{\text{earned}} = v_{\text{earned}}(G)$, and $v_{\text{invalid}} = v_{\text{invalid}}(G)$.

Every weakly connected component of G contains a simple path starting at an **Issue** node that contains all *valid* **Deduct** nodes within that component and no *invalid* **Deduct** nodes (Lemma 38). We obtain the subgraph G' of G as the (disjoint) union of these paths (one for each weakly connected component). As we have removed all invalid **Deduct** nodes but preserved all valid ones, we have $v_{\text{spent}}(G') = v_{\text{spent}}(G) - v_{\text{invalid}}(G)$. Because every weakly connected component G'' in G' is a path starting at an **Issue** node, we have that $v_{\text{spent}}(G'') \leq v_{\text{earned}}(G'')$ (Lemma 46). Because this holds for every weakly connected component G'' of G', we have $v_{\text{spent}}(G') \leq v_{\text{earned}}(G')$. Also, obviously $v_{\text{earned}}(G') \leq v_{\text{earned}}(G)$ by the subgraph property.

Overall, $v_{\text{spent}}(G) - v_{\text{invalid}}(G) = v_{\text{spent}}(G') \leq v_{\text{earned}}(G') \leq v_{\text{earned}}(G)$ □

Lemma 38. Every weakly connected component of G contains a simple path containing all valid **Deduct** nodes within that component and no invalid **Deduct** nodes.

Proof. Let G' be a weakly connected component in G. By Lemma 41, G' contains a single **Issue** node. Let j be the numerically largest index such that $j \in G'$ is a valid **Deduct** node (if no such j exists, the lemma's statement holds trivially). Because of Lemma 41, there exists a path

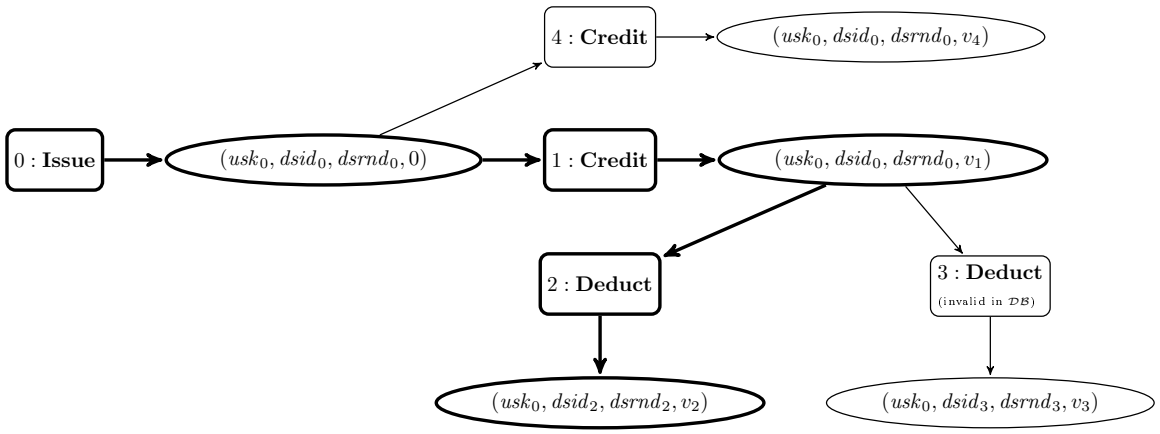

Figure 6: Example explanation graph G as in Lemma 35 (but with only one user).
The bold graph elements form the "canonical" path (Lemma 38) containing all valid deduct operations; all other nodes are removed in Lemma 37, ensuring $v_{\text{spent}}(G') = v_{\text{spent}}(G) - v_{\text{invalid}}(G)$ and $v_{\text{earned}}(G') \leq v_{\text{earned}}(G)$.

P from the **Issue** node to j. We show that P contains all valid **Deduct** nodes and no invalid nodes.

Assume for contradiction that P contains an invalid node. Then j would be invalid as well (Lemma 42), as it is reachable from that invalid node. Hence P does not contain invalid nodes.

Assume for contradiction that some **Deduct** node j' is valid but not on P. j' is reachable from the **Issue** node (Lemma 41) via some path P'. P and P' start at the same node but j is not on P' (because $j' < j$ by maximal choice of j and operation indices are monotonously increasing on any path (Lemma 39). Because j' is not on P and j is not on P', neither path is a prefix of the other, so there exists a node that differs on the two paths. Let $\vec{A}$ be the last node on P and P' *before* the first node that differs (note that this must be an attribute vector node as the operation nodes have out-degree 1 by definition). Let i be the first **Deduct** node after $\vec{A}$ on P and let i' be the first **Deduct** node after $\vec{A}$ on P'. Because i and i' are the first **Deduct** operation on each path (i.e. only **Credit** operations happen between $\vec{A}$ and i or i'), we have that $dsid_i = dsid_{i'}$ (where $dsid_\ell$ is the *dsid* that was revealed during the ℓth query). From the definition of DBsync, it is easy to see that i or i' must have been marked invalid (at most one transaction per *dsid* is valid). Since i is on P, it is valid. Hence i' must be invalid. Because j' is reachable from i' (via P'), j' must be invalid (Lemma 42) □

Lemma 39. For any path in G, the indices of **Issue** and **Deduct** nodes on the path are strictly monotonously increasing.

Proof. Let P be a path and let j and i be **Deduct** nodes on the path in that order (in the following, j could also be an **Issue** node) so that there are no other **Deduct** nodes between them on the path. Assume for contradiction that $i \leq j$. Let $\psi_i, \vec{A}_i = (usk_i, dsid_i, dsrnd_i, v_i), \alpha_i$ be the "input" values for query i (as defined by $\mathcal{L}$). Because $\mathcal{L}$ is consistent, there is some **Issue** or **Deduct** node $i' < i$ that creates attribute vectors with $dsid_i$. However, there is a path from j to i that involves only **Credit** and attribute vector nodes. This implies that the *dsid* in j's successor node is $dsid_i$. This means that $j \neq i'$ are associated with the same *dsid*, contradicting ¬error (cf. Lemma 35). □

Lemma 40. G is acyclic.

Proof. Assume there exists a cycle C. C cannot contain **Issue** nodes as they have in-degree 0. Because of Lemma 39, C cannot not contain any **Deduct** nodes. This means that the only oracle nodes on the cycle are **Credit** nodes. In turn, this implies that all $\vec{A} = (usk, dsid, dsrnd, v)$ nodes on the cycle share the same *usk*, *dsid*, *dsrnd* (as those are not changed by **Credit**). **Credit** strictly increases v, but on a cycle we would have to see a **Credit** node that decreases v or leaves it unchanged. Hence there are also no **Credit** on the cycle. Overall, there are only attribute vector nodes on the cycle, but there are no edges between attribute vector nodes, contradicting the existence of the cycle. □

Lemma 41. Every weakly connected component of G contains exactly one **Issue** node. Furthermore, every node in G is reachable from (exactly one) **Issue** node.

Proof. Let v be a node in G. Because G is acyclic (Lemma 40), the process of walking edges backwards from v eventually stops. It cannot stop at an attribute vector node (since every attribute vector node has in-degree at least 1 by consistency of $\mathcal{L}$) and it cannot stop at a **Credit** or **Deduct** node (as those have in-degree 1), hence it must stop at an **Issue** node. So v can be reached from some **Issue** node.

Assume for contradiction that some weakly connected component contains two **Issue** nodes v_0, v_1. By choice of our update functions, all attribute vector nodes $\vec{A} = (usk, dsid, dsrnd, v)$ reachable from a **Issue** node have the same *usk* (because no update ever changes *usk*). Furthermore, there are no two **Issue** nodes with the same *usk* (since **Issue**(*upk*) can only be called once

per *upk* and ComputePK is injective). As a consequence, every node is reachable from exactly one **Issue** node.

If we partition the attribute vector nodes in the weakly connected component into those that are reachable from v_0 and those that are reachable from v_1, there must be some path (of the form $\vec{A}_0 \to i \to \vec{A}_1$) from some $\vec{A}_0$ reachable from v_0 to some $\vec{A}_1$ reachable from v_1 or vice versa (otherwise the graph cannot be weakly connected). However, then $\vec{A}_1$ (or $\vec{A}_0$) is reachable from v_0 *and* from v_1, contradicting our previous result. This implies that every weakly connected component contains at most one **Issue** node. Furthermore, every weakly connected component contains at least one node, which is reachable from some **Issue**, meaning that it also contains *at least* one **Issue** node. □

Lemma 42. If **DBsync**(s) has been queried for all spend handles s, then every **Deduct** node that is reachable from an invalid **Deduct** node is invalid.

Proof. Let i, j be **Deduct** nodes such that i is invalid and j is reachable from i with no further **Deduct** nodes on the path P between i and j. If we can show that j is invalid, transitivity implies the statement for *all* j' reachable from i.

Because P does not contain (intermediate) **Deduct** nodes, all attribute vector nodes on P have the same *usk*, *dsid*, *dsrnd*. *dsid* is input to j's oracle query. Let $dstag_i = (c, \gamma, ctrace)$ be the double-spend tag output by Deduct in query i. Because of the update function used in query i, it holds that $\mathsf{Decrypt}_{\mathcal{E}}(pp, usk, ctrace) = dsid$.

We distinguish two cases: t_i is marked invalid before t_j was added to $\mathcal{DB}$ or vice versa. Assume t_i was marked invalid *before* t_j was added to the graph. When t_i was marked invalid, the successor node *dsid* is added to $\mathcal{DB}$ (Lemma 43). When t_j is added to the database afterwards, its input *dsid* is already in the database, hence t_j is immediately marked invalid. Assume that t_i was marked invalid *after* t_j was added to $\mathcal{DB}$. When t_j is added, *dsid* and an edge $(dsid, t_j)$ is added to $\mathcal{DB}$. Afterwards, at some point t_i is marked invalid. During this process, the edge $(t_i, dsid)$ is added to $\mathcal{DB}$ (Lemma 43) and because $(dsid, t_j)$ is in the graph, t_j is marked invalid.

Hence in both cases, t_j is marked invalid at some point. □

Lemma 43. Let t_i be some transaction node in $\mathcal{DB}$ and let i be the corresponding **Deduct** node in G with successor $\vec{A}^* = (usk^*, dsid^*, dsrnd^*, \cdot)$. After t_i is marked invalid, the successor of t_i in $\mathcal{DB}$ is $dsid^*$.

Proof. Since t_i is marked invalid, t_i's predecessor *dsid* in $\mathcal{DB}$ is correctly associated with some $(upk, dslink)$ (Lemma 45). In particular, i's predecessor $\vec{A} = (usk, dsid, dsrnd, v)$ in G must have $dslink = usk$. Let $\vec{A}^* = (usk^*, dsid^*, dsrnd^*, v^*)$ be i's successor. Let $dstag = (c, \gamma, ctrace)$ be the *dstag* associated with i's oracle query. Because of consistency of $\mathcal{L}$, we have $\mathsf{Decrypt}_{\mathcal{E}}(pp, usk, ctrace) = dsid^*$. When t_i is marked invalid, DBsync computes $dsid^* = \mathsf{Trace}(pp, dslink, dstag) = \mathsf{Decrypt}_{\mathcal{E}}(pp, usk, ctrace) = dsid^*$ and makes $dsid^*$ the successor to t_i. □

Lemma 44. For any two attribute vectors $\vec{A}_0 = (usk_0, dsid_0, dsrnd_0, v_0)$ and $\vec{A}_1 = (usk_1, dsid_1, dsrnd_1, v_1)$ in G, it holds that if $dsid_0 = dsid_1$, then $usk_0 = usk_1$ and $dsrnd_0 = dsrnd_1$.

Proof. Because of ¬error, there is a unique **Issue** or **Deduct** node i whose successor $\vec{A}^* = (usk^*, dsid^*, dsrnd^*, v_0)$ contains $dsid^* = dsid_0 = dsid_1$. Because i is unique in this regard, both $\vec{A}_0$ and $\vec{A}_1$ are reachable from i on paths P_0, P_1 that contains only **Credit** and attribute vector nodes. Since **Credit** does not change *usk* or *dsrnd*, we get that $usk_0 = usk_1 = usk^*$ and $dsrnd_0 = dsrnd_1 = dsrnd^*$. □

Lemma 45. We say that a node *dsid* in $\mathcal{DB}$ is "correctly associated" with $(upk, dslink)$ if there exists $(usk', dsid', dsrnd', \cdot)$ in G with $dsid'$ and $dslink = usk'$ and for all $(usk', dsid', dsrnd', \cdot)$ in G with $dsid'$, we have that $dslink = usk'$. All nodes *dsid* in $\mathcal{DB}$ that have some value associated with them are correctly associated.

Proof. Let *dsid* be some node in $\mathcal{DB}$ that has been associated with some value $(upk, dslink)$. We prove the claim essentially via induction: We first show that if $(upk, dslink)$ was computed when adding a second transaction to *dsid* to $\mathcal{DB}$, then it is correctly associated. We then show that if *dsid* has been correctly associated with $(upk, dslink)$, then copying the value to some $dsid^*$ in the "when t_i is marked invalid" trigger correctly associated $(upk, dslink)$ to $dsid^*$. In each case, it suffices to show that $dslink = usk$ for *some* $(usk, dsid, dsrnd, \cdot)$ in G, as Lemma 44 then implies that this holds for all attribute vector nodes with *dsid*.

To show the first statement, let t_i be a transaction node in $\mathcal{DB}$ with predecessor *dsid* and assume t_j with the same predecessor is added to $\mathcal{DB}$ by DBsync. Let $dstag_i, dstag_j$ be their *dstag*s. Let i, j be the **Deduct** nodes in G corresponding to t_i and t_j, respectively[3]. Let $\vec{A}_i = (usk_i, dsid_i, dsrnd_i, v_i)$ and $\vec{A}_j = (usk_j, dsid_j, dsrnd_j, v_j)$ be the predecessors of i and j in G, respectively. It holds that $dsid_i = dsid_j = dsid$ by consistency of $\mathcal{L}$ (since equality with *dsid* is checked by the update function). Because $dsid_i = dsid_j$, we get $usk_i = usk_j$ and $dsrnd_i = dsrnd_j$ (Lemma 44). Because of consistency of $\mathcal{L}$, necessarily $dstag_i = (c_i = usk_i \cdot \gamma_i + dsrnd_i, \gamma_i, ctrace)$ and $dstag_j = (c_j = usk_j \cdot \gamma_j + dsrnd_j, \gamma_j, ctrace)$ (as enforced by the update function). Since $usk_i = usk_j$ and $dsrnd_i = dsrnd_j$ and $\gamma_i \neq \gamma_j$ (as implied by $\neg$error), we get $dslink = (c_i - c_j)/(\gamma_i - \gamma_j) = usk_i$. Hence $dslink = usk_i$ for our attribute vector $(usk_i, dsid_i, dsrnd_i, v_i)$ in G.

To show the second statement, let t_i be a transaction that is marked invalid. Let *dsid* be its predecessor (which is by assumption correctly associated with $(upk, dslink)$). Let $dstag = (c, \gamma, ctrace)$ be the associated *dstag* for t_i. Let $dsid^* = \mathsf{Trace}(ispp, dslink, dstag) = \mathsf{Decrypt}_{\mathcal{E}}(pp, dslink, ctrace)$ be t_i's successor. We show that $dsid^*$ is correctly associated with $(upk^*, dslink^*)$. Let $(usk, dsid, dsrnd, \cdot)$ be the predecessor of i in G. By assumption it *dsid* is correctly associated, hence $usk = dslink$. Let $(usk', dsid', dsrnd', \cdot)$ be the successor of i in G. By consistency of $\mathcal{L}$, $\mathsf{Decrypt}_{\mathcal{E}}(pp, usk = dslink, ctrace) = dsid'$ as guaranteed by the update function ψ. Hence $dsid' = dsid^*$. Because $usk = usk' = dslink$, we have that $(usk, dsid^*, dsrnd', \cdot)$ in G contains $dsid^*$ and $dslink = usk$, implying that $dsid^*$ is correctly associated with $(upk, dslink)$. □

Lemma 46. On every path P in G starting at some **Issue** node, it holds that $v_{\text{spent}}(P) \leq v_{\text{earned}}(P)$.

Proof. Let $(usk, dsid, dsrnd, v)$ be the successor (in G) of the last **Deduct** node on P. By design of our update functions, it is easy to see that $v \leq \sum_{i \in P \text{ is } \textbf{Credit} \text{ node}} k_i - \sum_{j \in P \text{ is } \textbf{Deduct} \text{ node}} k_j = v_{\text{earned}}(P) - v_{\text{spent}}(P)$. (the inequality is usually an equality, assuming that there is no **Credit** operation where adding k to the current v exceeds $v_{max} = p - 1$. If the latter happens, the integers will wrap around and result in the smaller $v' = v + k \mod p$) Also by design of the update functions, it holds that $v \geq 0$. Hence $v_{\text{earned}}(P) - v_{\text{spent}}(P) \geq 0$. □

D.4 Incentive System Framing Resistance

Theorem 17. Let $\mathcal{A}$ be a ppt adversary against framing resistance of our incentive system. Without loss of generality, we assume that $\mathcal{A}$ always outputs some actual user's handle u in the challenge phase. Let k be a (polynomial in λ) upper bound for the number of **Keygen** calls that $\mathcal{A}$ may make. We construct $\mathcal{B}$ against CPA-security of $\Pi_{\mathcal{E}}$.

- $\mathcal{B}$ gets pp, pk^* from its challenger. It finishes the incentive system setup by simulating the UACS setup $cpp \leftarrow \mathfrak{S}_{\mathsf{Setup}}(pp)$ and computing $pk_{\mathcal{C}^+}$ as usual. It hands $ispp = (pp, cpp, pk_{\mathcal{C}^+})$ to $\mathcal{A}$.
- $\mathcal{B}$ randomly chooses an index $j \leq k$. For the jth **Keygen** query, $\mathcal{B}$ responds with $upk = pk^*$ and some handle u^*.

[3]This is a slight abuse of notation as the index i of t_i does not necessarily correspond to node i in G, which is associated with the ith oracle query

- Any queries involving u^* are run honestly by $\mathcal{B}$ except that it uses the UACS simulators to simulate the $\mathsf{Receive}$ and Update protocols without usk.
- Eventually, $\mathcal{A}$ enters the challenge phase and outputs some $dslink$ and a user handle u.
- If $u \neq u^*$, $\mathcal{B}$ aborts.
- If $upk^* \neq \mathsf{ComputePK}_{\mathcal{E}}(pp, dslink)$, $\mathcal{B}$ aborts.
- Otherwise, $\mathcal{B}$ uses $dslink$ as the secret key to pk^* to break its CPA challenge with probability 1.

The view of $\mathcal{A}$ in the framing resistance game is simulated perfectly and independently of j. We have that $\Pr[B \text{ wins the CPA game}] = \Pr[\text{Exp}^{\text{frame-res}}(\Pi, \mathcal{A}, \lambda) = 1] \cdot \Pr[u = u^*]$. By assumption, $\Pi_{\mathcal{E}}$ is CPA-secure. $\Pr[u = u^*]$ is non-negligible, hence $\Pr[\text{Exp}^{\text{frame-res}}(\Pi, \mathcal{A}, \lambda) = 1]$ must be negligible. $\square$